

Undergraduate Topics in Computer Science

Wolfgang Ertel

Introduction to Artificial Intelligence



 Springer

Bachelor-Themen in Informatik

Undergraduate Topics in Computer Science (UTiCS) bietet hochwertige Lehrinhalte für Studenten aller Bereiche der Informatik und Informationswissenschaft. Von grundlegendem und theoretischem Material bis hin zu Abschlussthemen und -anwendungen verfolgen UTiCS-Bücher einen frischen, prägnanten und modernen Ansatz und eignen sich ideal für das Selbststudium oder für einen ein- oder zweisemestrigen Kurs. Die Texte sind alle von ausgewiesenen Experten auf ihrem Gebiet verfasst, von einem internationalen Beirat geprüft und enthalten zahlreiche Beispiele und Problemstellungen. Viele enthalten vollständig funktionierende Lösungen.

Für weitere Bände:

www.springer.com/series/7592

Wolfgang Ertel

Einführung in Künstliches Intelligenz

**Übersetzt von Nathanael Black
Mit Illustrationen von Florian Mast**

 **Springer**

Prof. Dr. Wolfgang Ertel
FB Elektrotechnik und Informatik
Hochschule Ravensburg-Weingarten
Fachhochschule Weingarten
Deutschland
[ertel@hs-
weingarten.de](mailto:ertel@hs-weingarten.de)

Serieneditor
Ian Mackie

Beirat

Samson Abramsky, Universität Oxford, Oxford, Großbritannien
Karin Breitman, Päpstliche Katholische Universität Rio de Janeiro, Rio de Janeiro, Brasilien
Chris Hankin, Imperial College London, London, Großbritannien
Dexter Kozen, Cornell University, Ithaca, USA
Andrew Pitts, Universität Cambridge, Cambridge, Großbritannien
Hanne Riis Nielson, Technische Universität Dänemark, Kongens Lyngby, Dänemark
Steven Skiena, Stony Brook University, Stony Brook, USA
Iain Stewart, University of Durham, Durham, Großbritannien

ISSN 1863-7310

ISBN 978-0-85729-298-8

e-ISBN 978-0-85729-299-5

DOI 10.1007/978-0-85729-299-5

Springer London Dordrecht Heidelberg New York

Katalogisierung der British Library in Publikationsdaten
Ein Katalogeintrag für dieses Buch ist in der British Library erhältlich

Kontrollnummer der Library of Congress: 2011923216

Ursprünglich in deutscher Sprache veröffentlicht von Vieweg+Teubner, 65189 Wiesbaden, Deutschland, als „Ertel, Wolfgang: Grundkurs Künstliche Intelligenz. 2. rev. Auflage“. © Vieweg+Teubner
| GWV Fachverlage GmbH, Wiesbaden 2009

© Springer-Verlag London Limited 2011

Abgesehen von fairem Handel zum Zwecke der Forschung oder des privaten Studiums oder der Kritik oder Rezension gemäß dem Copyright, Designs and Patents Act 1988 darf diese Veröffentlichung nur reproduziert, gespeichert oder übertragen werden, in irgendeiner Form oder mit irgendwelchen Mitteln, mit vorheriger schriftlicher Genehmigung des Herausgebers oder im Falle einer reprografischen Vervielfältigung gemäß den von der Copyright Licensing Agency ausgestellten Lizenzbedingungen. Anfragen bezüglich einer Vervielfältigung außerhalb dieser Bedingungen sind an den Verlag zu richten.

Die Verwendung eingetragener Namen, Marken usw. in dieser Veröffentlichung bedeutet, auch wenn keine ausdrückliche Angabe erfolgt, nicht, dass diese Namen von den einschlägigen Gesetzen und Vorschriften ausgenommen und daher für die allgemeine Verwendung frei sind.

Der Herausgeber gibt keine Zusicherung, weder ausdrücklich noch stillschweigend, hinsichtlich der Richtigkeit der in diesem Buch enthaltenen Informationen und kann keine rechtliche Verantwortung oder Haftung für etwaige Fehler oder Auslassungen übernehmen.

Gedruckt auf säurefreiem Papier

Springer ist Teil von Springer Science+Business Media (www.springer.com)

Vorwort

Künstliche Intelligenz (KI) hat das klare Ziel, Intelligenz zu verstehen und intelligente Systeme aufzubauen. Allerdings sind die Methoden und Formalismen auf dem Weg zu diesem Ziel nicht fest verankert, was dazu geführt hat, dass KI heute aus einer Vielzahl von Teildisziplinen besteht. Die Schwierigkeit bei einem Einführungskurs in die KI liegt darin, möglichst viele Zweige zu vermitteln, ohne zu viel Tiefe und Präzision zu verlieren.

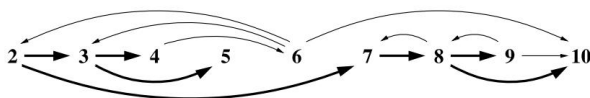
Das Buch von Russell und Norvig [RN10] ist mehr oder weniger die Standardeinführung in die KI. Da dieses Buch jedoch 1.152 Seiten umfasst und für die meisten Studenten zu umfangreich und zu teuer ist, waren die Anforderungen für das Schreiben dieses Buches klar: Es sollte eine leicht zugängliche Einführung in die moderne KI zum Selbststudium oder als Grundlage für ein Viererstudium sein. Eine einstündige Vorlesung mit höchstens 300 Seiten. Das Ergebnis liegt vor.

Auf 300 Seiten kann ein so umfangreiches Fachgebiet wie KI nicht vollständig abgedeckt werden. Um zu vermeiden, dass das Buch zu einem Inhaltsverzeichnis wird, habe ich versucht, in die Tiefe zu gehen und konkrete Algorithmen und Anwendungen in jedem der folgenden Zweige vorzustellen: Agenten, Logik, Suche, Argumentation mit Unsicherheit, maschinelles Lernen und neuronale Netze.

Die Bereiche Bildverarbeitung, Fuzzy-Logik und Verarbeitung natürlicher Sprache werden nicht im Detail behandelt. Der für die gesamte Informatik wichtige Bereich der Bildverarbeitung ist eine eigenständige Disziplin mit sehr guten Lehrbüchern, wie z. B. [GW08]. Einen ähnlichen Status hat die Verarbeitung natürlicher Sprache. Bei der Erkennung und Generierung von Text und gesprochener Sprache werden Methoden der Logik, des probabilistischen Denkens und neuronaler Netze angewendet. In diesem Sinne ist dieser Bereich Teil der KI. Andererseits ist die Computerlinguistik ein eigener umfangreicher Zweig der Informatik und hat viele Gemeinsamkeiten mit formalen Sprachen. In diesem Buch werden wir an mehreren Stellen auf solche geeigneten Systeme hinweisen, jedoch keine systematische Einführung geben. Für eine erste Einführung in dieses Gebiet verweisen wir auf die Kapitel 22 und 23 in [RN10]. Die Fuzzy-Logik bzw. Fuzzy-Set-Theorie hat sich aufgrund ihrer primären Anwendung in der Automatisierungstechnik zu einem Teilgebiet der Regelungstheorie entwickelt und wird in den entsprechenden Büchern behandelt. Daher verzichten wir hier auf eine Einführung.

Die Abhängigkeiten zwischen den Kapiteln des Buches sind in der unten gezeigten Grafik grob skizziert. Um es einfach zu halten, Kap. 1 mit der grundsätzlichen Einleitung für alle weiteren Kapitel entfällt. Der dickere Pfeil von 2 nach 3 bedeutet beispielsweise, dass die Aussagenlogik eine Voraussetzung für das Verständnis der Prädikatenlogik ist. Der dünne Pfeil von 9 bis 10 bedeutet, dass neuronale Netze für das Verständnis von Reinforcement Learning hilfreich, aber nicht unbedingt notwendig sind. Dünne Pfeile nach hinten

Pfeile sollen deutlich machen, dass spätere Kapitel das Verständnis vertiefen können zu bereits erlernten Themen.



Dieses Buch richtet sich an Studierende der Informatik und anderer technischer Naturwissenschaften und erfordert größtenteils Mathematikkenntnisse auf High-School-Niveau. An mehreren Stellen Erkenntnisse aus der linearen Algebra und der mehrdimensionalen Analysis wird gebraucht. Für ein tieferes Verständnis der Inhalte ist die aktive Bearbeitung der Übungen unabdingbar. Das bedeutet, dass die Lösungen nur konsultiert werden sollten nach intensiver Arbeit mit jedem Problem, und nur um die eigenen Lösungen zu überprüfen, getreu Leonardo da Vincis Motto „Lernen ohne Hingabe schadet dem Gehirn“. Etwas Schwierigere Probleme sind mit ♪ und besonders schwierige mit ♪ ♪ gekennzeichnet. Probleme, die Programmierkenntnisse oder spezielle Informatikkenntnisse erfordern, sind mit ♪ gekennzeichnet.

Auf der Website des Buches unter www.hs-weingarten.de/~ertel/aibook digitale Materialien zu den Übungen wie Trainingsdaten für Lernalgorithmen, eine Seite mit Verweisen auf im Buch erwähnte KI-Programme, eine Linkliste zu die behandelten Themen, a anklickbare Liste der Bibliographie, eine Errata-Liste und Präsentationsfolien für Dozenten kann gefunden werden. Ich bitte den Leser, Anregungen, Kritik und Tipps dazu zu senden Fehler direkt an ertel@hs-weingarten.de.

Dieses Buch ist eine aktualisierte Übersetzung meines deutschen Buches „Grundkurs Künstliche Intelligenz“ erschienen im Vieweg Verlag. Mein besonderer Dank gilt dem Übersetzer Nathan Black, der in einer hervorragenden transatlantischen Zusammenarbeit zwischen Deutschland und Kalifornien hat über SVN, Skype und E-Mail diesen Text erstellt. Ich bin Franz Kurfuß dankbar, der mich Nathan vorgestellt hat; an Matthew Wight für das Korrekturlesen der Übersetzung Buch und an Simon Rees vom Springer Verlag für seine Geduld.

Ich möchte meiner Frau Evelyn für ihre Unterstützung und Geduld dabei danken zeitaufwändiges Projekt. Besonderer Dank gilt Wolfgang Bibel und Chris Lobenschuss, die das deutsche Manuskript sorgfältig korrigiert haben. Ihre Anregungen und Diskussionen führen zu vielen Verbesserungen und Ergänzungen. Zum Lesen der Korrekturen und Für weitere wertvolle Dienste möchte ich mich bei Richard Cubek, Celal Döven und Joachim bedanken Feßler, Nico Hochgeschwender, Paul Kirner, Wilfried Meister, Norbert Perk, Peter Radtke, Markus Schneider, Manfred Schramm, Uli Stärk, Michel Tokic, Arne Usadel und alle interessierten Studierenden. Mein Dank geht auch an Florian Mast für die Unbezahlbare Cartoons und eine sehr effektive Zusammenarbeit.

Ich hoffe, dass dieses Buch Ihnen während Ihres Studiums dabei hilft, meine Faszination mit Ihnen zu teilen Künstliche Intelligenz.

Inhalt

1 Einleitung	1
1.1 Was ist künstliche Intelligenz?	1
1.1.1 Gehirnforschung und Problemlösung.	3
1.1.2 Der Turing-Test und Chatterbots . 4	4
1.2 Die Geschichte der KI. 5	5
1.2.1 Die ersten Anfänge . . 5	5
1.2.2 Logik löst (fast) alle Probleme 1.2.3	6
Der neue Konnektionismus	7
1.2.4 Argumentation unter Unsicherheit	7
1.2.5 Verteilte, autonome und lernende Agenten	8
1.2.6 KI wird erwachsen. 9	9
1.3 Agenten	9
1.4 Wissensbasierte Systeme . 12	12
1.5 Übungen 14	14
2 Aussagenlogik	15
2.1 Syntax	15
2.2 Semantik	16
2.3 Beweissysteme	18
2.4 Auflösung	22
2.5 Hornklauseln	25
2.6 Berechenbarkeit und Komplexität . 2.7	28
Anwendungen und Einschränkungen	29
2.8 Übungen	29
3 Prädikatenlogik erster Ordnung	31
3.1 Syntax	32
3.2 Semantik	33
3.2.1 Gleichheit	36
3.3 Quantoren und Normalformen	37
3.4 Beweiskalküle	40
3.5 Auflösung	42
3.5.1 Lösungsstrategien	46
3.5.2 Gleichheit	46
3.6 Automatisierte Theorembeweiser.. . . .	47

3.7 Mathematische Beispiele . . . 3.8	48
Anwendungen	51
3.9 Zusammenfassung	54
3.10 Übungen	54
4 Grenzen der Logik	57
4.1 Das Suchraumproblem 4.2	57
Entscheidbarkeit und Unvollständigkeit	59
4.3 Der fliegende Pinguin	60
4.4 Modellierungsunsicherheit 4.5	63
Übungen	65
5 Logikprogrammierung mit PROLOG	67
5.1 PROLOG-Systeme und Implementierungen	68
5.2 Einfache Beispiele	68
5.3 Ausführungskontrolle und Verfahrenselemente	71
5.4 Listen	73
5.5 Selbstmodifizierende Programme	75
5.6 Ein Planungsbeispiel 5.7 5.8	76
Constraint-Logik-Programmierung	77
Zusammenfassung	79
5.9 Übungen	80
6 Suche, Spiele und Problemlösung . 6.1	83
Einführung 6.2	83
Uninformierte Suche	89
6.2.1 Breitensuche	89
6.2.2 Tiefensuche	91
6.2.3 Iterative Vertiefung	92
6.2.4 Vergleich	94
6.3 Heuristische Suche.	94
6.3.1 Gierige Suche	96
6.3.2 A* -Suche	98
6.3.3 IDA* -Suche	100
6.3.4 Empirischer Vergleich der Suchalgorithmen	100
6.3.5 Zusammenfassung	102
6.4 Spiele mit Gegnern	102
6.4.1 Minimax-Suche	103
6.4.2 Alpha-Beta-Pruning	103
6.4.3 Nichtdeterministische Spiele	106
6.5 Heuristische	106
Auswertungsfunktionen 6.5.1	107
6.6 Erlernen von	108
6.7 Heuristiken Stand der Technik Übungen	110
7 Argumentieren mit Unsicherheit.	113
7.1 Rechnen mit Wahrscheinlichkeiten	115

7.1.1 Bedingte Wahrscheinlichkeit	118
7.2 Das Prinzip der maximalen Entropie	122
7.2.1 Eine Inferenzregel für	123
Wahrscheinlichkeiten 7.2.2 Maximale Entropie ohne explizite Einschränkungen	127
7.2.3 Bedingte Wahrscheinlichkeit versus materielle Implikation	128
7.2.4 MaxEnt-Systeme	129
7.2.5 Das Tweety-Beispiel	130
7.3 LEXMED, ein Expertensystem zur Diagnose von Blinddarmentzündungen	131
7.3.1 Blinddarmentzündungsdiagnose mit formalen	131
Methoden 7.3.2 Hybride probabilistische	132
Wissensbasis 7.3.3 Anwendung	135
von LEXMED 7.3.4 Funktion	136
von LEXMED 7.3.5 Risikomanagement mithilfe	139
der Kostenmatrix	141
7.3.6 Leistung 7.3.7 Anwendungsbereiche und Erfahrungen	143
7.4 Argumentation mit Bayes'schen Netzwerken	144
7.4.1 Unabhängige Variablen	144
7.4.2 Grafische Darstellung von Wissen als Bayesian Netzwerk.	146
7.4.3 Bedingte Unabhängigkeit	146
7.4.4 Praktische Anwendung	148
7.4.5 Software für Bayesianische Netzwerke	149
7.4.6 Entwicklung Bayes'scher Netzwerke	150
7.4.7 Semantik Bayes'scher Netzwerke	154
7.5 Zusammenfassung	156
7.6 Übungen	157
8 Maschinelles Lernen und Data Mining	161
8.1 Datenanalyse	166
8.2 Das Perzeptron, ein linearer Klassifikator	169
8.2.1 Die Lernregel 8.2.2	171
Optimierung und Ausblick	174
8.3 Die Nearest-Neighbor-Methode	175
8.3.1 Zwei Klassen, viele Klassen, Approximation	179
8.3.2 Entfernung ist relevant	180
8.3.3 Rechenzeiten 8.3.4	181
Zusammenfassung und Ausblick	182
8.3.5 Fallbasiertes Denken	183
8.4 Entscheidungsbaum-Lernen	184
8.4.1 Ein einfaches Beispiel	185
8.4.2 Entropie als Maß für Informationsinhalte	186
8.4.3 Informationsgewinn	189
8.4.4 Anwendung von C4.5	191
8.4.5 Lernen der Blinddarmentzündungsdiagnose	193
8.4.6 Kontinuierliche Attribute	196

8.4.7 Beschneiden – Den Baum	197
8.4.8 Fehlende Werte	198
8.4.9 Zusammenfassung	199
8.5 Lernen von Bayes'schen Netzwerken	199
8.5.1 Erlernen der Netzwerkstruktur	199
8.6 Der Naive-Bayes-Klassifikator	202
8.6.1 Textklassifizierung mit Naive Bayes	204
8.7 Clustering	206
8.7.1 Distanzmetriken	207
8.7.2 k-Means und der EM-Algorithmus	208
8.7.3 Hierarchisches Clustering	209
8.8 Data Mining in der Praxis	211
8.8.1 Das Data Mining Tool KNIME	212
8.9 Zusammenfassung	214
8.10 Übungen	216
8.10.1 Einführung	216
8.10.2 Das Perzeptron	216
8.10.3 Nearest-Neighbor-Methode	217
8.10.4 Entscheidungsbäume	218
8.10.5 Lernen von Bayes'schen Netzwerken	219
8.10.6 Clustering	220
8.10.7 Data Mining	220
9 Neuronale Netze	221
9.1 Von der Biologie zur Simulation	222
9.1.1 Das mathematische Modell	223
9.2 Hopfield-Netzwerke	226
9.2.1 Anwendung auf ein Mustererkennungsbeispiel	227
9.2.2 Analyse	228
9.2.3 Zusammenfassung und Ausblick	231
9.3 Neuronales Assoziatives Gedächtnis	232
9.3.1 Korrelationsmatrixspeicher	233
9.3.2 Die Pseudoinverse	235
9.3.3 Die binäre Hebb-Regel	236
9.3.4 Ein Programm zur Rechtschreibkorrektur	238
9.4 Lineare Netzwerke mit minimalen Fehlern	240
9.4.1 Methode der kleinsten Quadrate	241
9.4.2 Anwendung auf die Appendizitis-Daten	242
9.4.3 Die Delta-Regel	243
9.4.4 Vergleich mit dem Perzeptron	245
9.5 Der Backpropagation-Algorithmus	246
9.5.1 NETalk: Ein Netzwerk lernt sprechen	249
9.5.2 Erlernen von Heuristiken für Theorembeweiser	250
9.5.3 Probleme und Verbesserungen	251
9.6 Support Vector Machines	252

9.7 Anwendungen	254
9.8 Zusammenfassung und Ausblick . . 9.9	255
Übungen 9.9.1 Von der Biologie zur Simulation:	255
9.9.2 Hopfield-Netzwerke	255
9.9.3 Lineare Netzwerke mit minimalen	255
Fehlern 9.9.4 Backpropagation	256
9.9.5 Support Vector Machines	256
10 Verstärkungslernen.	257
10.1 Einführung 10.2	257
Die Aufgabe 10.3	259
Uninformierte kombinatorische Suche	261
10.4 Werteiteration und dynamische Programmierung :	262
10.5 Ein lernender Laufroboter und seine Simulation	265
10.6 Q-Learning	267
10.6.1 Q-Learning in einer nichtdeterministischen Umgebung.	270
10.7 Exploration und Ausbeutung	272
10.8 Approximation, Generalisierung und Konvergenz	272
10.9 Anwendungen	273
10.10 Fluch der Dimensionalität	274
10.11 Zusammenfassung und Ausblick	275
10.12 Übungen	276
11 Lösungen zu den Übungen	279
11.1 Einführung	279
11.2 Aussagenlogik	280
11.3 Prädikatenlogik erster Ordnung.	282
11.4 Grenzen der Logik	283
11.5 PROLOG	283
11.6 Suche, Spiele und Problemlösung	285
11.7 Argumentieren mit Unsicherheit.	287
11.8 Maschinelles Lernen und Data Mining	293
11.9 Neuronale Netze	300
11.10 Verstärkungslernen	301
Verweise	305
Index	311

1.1 Was ist künstliche Intelligenz?

Der Begriff *Künstliche Intelligenz* weckt Emotionen. Da ist zum einen unsere Faszination für *Intelligenz*, die uns Menschen scheinbar einen besonderen Platz unter den Lebensformen einräumt. Es stellen sich Fragen wie „*Was ist Intelligenz?*“, „*Wie kann man Intelligenz messen?*“ oder „*Wie funktioniert das Gehirn?*“. All diese Fragen sind sinnvoll, wenn es darum geht, künstliche Intelligenz zu verstehen. Die zentrale Frage für den Ingenieur, insbesondere für den Informatiker, ist jedoch die Frage nach der intelligenten Maschine, die sich wie ein Mensch verhält und intelligentes Verhalten zeigt.

Das Attribut „*künstlich*“ könnte ganz andere Assoziationen hervorrufen. Es weckt Ängste vor intelligenten Cyborgs. Es erinnert an Bilder aus Science-Fiction-Romanen. Es stellt sich die Frage, ob wir versuchen sollten, unser höchstes Gut, die Seele, zu verstehen, zu modellieren oder sogar zu rekonstruieren.

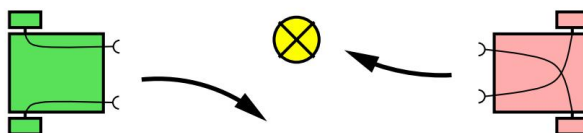
Bei solch unterschiedlichen Interpretationen wird es schwierig, den Begriff *künstliche Intelligenz* oder *KI* einfach und eindeutig zu definieren. Dennoch möchte ich versuchen, anhand von Beispielen und historischen Definitionen den Bereich der KI zu charakterisieren. Im Jahr 1955 definierte John McCarthy, einer der Pioniere der KI, als erster den Begriff *künstliche Intelligenz* etwa wie folgt:

Ziel der KI ist es, Maschinen zu entwickeln, die sich so verhalten, als wären sie intelligent.

Um diese Definition zu testen, könnte sich der Leser das folgende Szenario vorstellen. Etwa fünfzehn kleine Roboterfahrzeuge bewegen sich auf einer geschlossenen, vier mal vier Meter großen Fläche. Man kann verschiedene Verhaltensmuster beobachten. Manche Fahrzeuge bilden kleine Gruppen mit relativ wenig Bewegung. Andere bewegen sich friedlich durch den Raum und weichen jeder Kollision elegant aus. Wieder andere scheinen einem Anführer zu folgen. Auch aggressives Verhalten ist zu beobachten. Ist das, was wir sehen, intelligent?

Nach McCarthys Definition können die oben genannten Roboter als intelligent bezeichnet werden. Der Psychologe Valentin Braitenberg hat gezeigt, dass dieses scheinbar komplexe Verhalten durch sehr einfache elektrische Schaltkreise hervorgerufen werden kann [Bra84]. Sogenannte Braitenberg-Fahrzeuge verfügen über zwei Räder, die jeweils von einem unabhängigen Elektromotor angetrieben werden. Die Geschwindigkeit jedes Motors wird durch eine

Abb. 1.1 Zwei sehr einfache
Braitenberg-Fahrzeuge und ihre
Reaktionen auf eine Lichtquelle



vorne am Fahrzeug, wie in Abb. 1.1 dargestellt. Je mehr Licht auf den Sensor trifft, desto schneller läuft der Motor. Fahrzeug 1 im linken Bildteil entfernt sich entsprechend seiner Konfiguration von einer Punktlichtquelle. Fahrzeug 2 hingegen bewegt sich auf die Lichtquelle zu. Durch weitere kleine Modifikationen können andere Verhaltensmuster entstehen, sodass wir mit diesen sehr einfachen Fahrzeugen das oben beschriebene beeindruckende Verhalten realisieren können.

Offensichtlich ist die obige Definition unzureichend, da KI das Ziel hat, schwierige praktische Probleme zu lösen, die für das Braitenberg-Fahrzeug sicherlich zu anspruchsvoll sind. In der Encyclopedia Britannica [Bri91] findet man eine Definition, die wie folgt lautet:

KI ist die Fähigkeit digitaler Computer oder computergesteuerter Roboter, Probleme zu lösen, die normalerweise mit den höheren intellektuellen Verarbeitungsfähigkeiten des Menschen verbunden sind

Doch diese Definition weist auch Schwächen auf. Es würde beispielsweise zugeben, dass ein Computer mit großem Speicher, der einen langen Text speichern und bei Bedarf abrufen kann, über intelligente Fähigkeiten verfügt, denn das Auswendiglernen langer Texte kann sicherlich als eine *höhere intellektuelle Verarbeitungsfähigkeit* des Menschen angesehen werden, ebenso wie beispielsweise die schnelle Multiplikation zweier 20-stelliger Zahlen. Nach dieser Definition ist also jeder Computer ein KI-System. Dieses Dilemma wird durch die folgende Definition von Elaine Rich [Ric83] elegant gelöst:

Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, Dinge zu tun, in denen Menschen derzeit besser sind.

Rich, prägnant und prägnant charakterisiert, was KI-Forscher in den letzten 50 Jahren getan haben. Auch im Jahr 2050 wird diese Definition noch aktuell sein.

Aufgaben wie die Ausführung vieler Berechnungen in kurzer Zeit sind die Stärken digitaler Computer. In dieser Hinsicht übertreffen sie den Menschen um ein Vielfaches. In vielen anderen Bereichen ist der Mensch der Maschine jedoch weit überlegen. Wer beispielsweise einen fremden Raum betritt, erkennt die Umgebung innerhalb von Sekundenbruchteilen und kann bei Bedarf ebenso schnell Entscheidungen treffen und Maßnahmen planen. Bisher ist diese Aufgabe für autonome Roboter zu anspruchsvoll. Nach Richs Definition ist dies also eine Aufgabe der KI. Tatsächlich ist die Forschung zu autonomen Robotern ein wichtiges, aktuelles Thema in der KI. Der Bau von Schachcomputern hingegen hat an Bedeutung verloren, da diese bereits auf dem Niveau von Großmeistern oder darüber hinaus spielen.

Es wäre jedoch gefährlich, aus Richs Definition zu schließen, dass es bei KI nur um die pragmatische Umsetzung intelligenter Prozesse geht. Intelligente Systeme im Sinne von Richs Definition können nicht ohne eine tiefe Intelligenz aufgebaut werden

¹Ein autonomer Roboter arbeitet selbstständig, ohne manuelle Unterstützung, insbesondere ohne Fernbedienung Kontrolle.

Verständnis des menschlichen Denkens und intelligenten Handelns im Allgemeinen, weshalb die Neurowissenschaften (siehe Abschn. 1.1.1) für die KI von großer Bedeutung sind. Dies zeigt auch, dass die anderen zitierten Definitionen wichtige Aspekte der KI widerspiegeln.

Eine besondere Stärke der menschlichen Intelligenz ist die Anpassungsfähigkeit. Wir sind in der Lage, uns durch Lernen an verschiedene Umweltbedingungen anzupassen und unser Verhalten entsprechend zu ändern. Gerade weil unsere Lernfähigkeit der von Computern so weit überlegen ist, ist *maschinelles Lernen* nach Richs Definition ein zentrales Teilgebiet der KI.

1.1.1 Gehirnforschung und Problemlösung

Durch die Erforschung intelligenter Systeme können wir versuchen zu verstehen, wie das menschliche Gehirn funktioniert, und es dann am Computer modellieren oder simulieren. Viele Ideen und Prinzipien auf dem Gebiet der neuronalen Netze (siehe Kapitel 9) stammen aus der Hirnforschung und dem verwandten Gebiet der Neurowissenschaften.

Ein ganz anderer Ansatz ergibt sich daraus, zielorientiert vorzugehen, von einem Problem auszugehen und zu versuchen, die optimale Lösung zu finden. Wie der Mensch das Problem löst, wird hier als unwichtig behandelt. Die Methode ist bei diesem Ansatz zweitrangig. An erster Stelle steht die optimale intelligente Lösung des Problems. Anstatt eine feste Methode (wie zum Beispiel die Prädikatenlogik) anzuwenden, hat die KI stets das Ziel, intelligente Agenten für möglichst viele verschiedene Aufgaben zu schaffen. Da die Aufgaben sehr unterschiedlich sein können, ist es nicht verwunderlich, dass die derzeit in der KI eingesetzten Methoden oft auch sehr unterschiedlich sind. Ähnlich wie die Medizin, die viele verschiedene, oft lebensrettende Diagnose- und Therapieverfahren umfasst, bietet auch KI eine breite Palette wirksamer Lösungen für unterschiedlichste Anwendungsbereiche.

Betrachten Sie zur geistigen Inspiration Abb. 1.2 auf Seite 4. Genau wie in der Medizin gibt es keine universelle Methode für alle Anwendungsbereiche von KI, sondern eine Vielzahl möglicher Lösungen für die Vielzahl unterschiedlicher großer und kleiner Alltagsprobleme.

Die *Kognitionswissenschaft* widmet sich der Erforschung des menschlichen Denkens auf einem etwas höheren Niveau. Ähnlich wie die Hirnforschung liefert dieser Bereich der praktischen KI viele wichtige Ideen. Andererseits führen Algorithmen und Implementierungen zu weiteren wichtigen Schlussfolgerungen darüber, wie menschliches Denken funktioniert. Somit profitieren diese drei Bereiche von einem fruchtbaren interdisziplinären Austausch. Gegenstand dieses Buches ist jedoch in erster Linie die problemorientierte KI als Teildisziplin der Informatik.

Es gibt viele interessante philosophische Fragen rund um Intelligenz und künstliche Intelligenz. Wir Menschen haben Bewusstsein; das heißt, wir können über uns selbst nachdenken und sogar darüber nachdenken, dass wir in der Lage sind, über uns selbst nachzudenken. Wie entsteht Bewusstsein? Viele Philosophen und Neurologen glauben heute, dass Geist und Bewusstsein mit der Materie, also mit dem Gehirn, verbunden sind. Die Frage, ob Maschinen eines Tages einen Geist oder ein Bewusstsein haben könnten, könnte irgendwann in der Zukunft relevant werden. Beim Geist-Körper-Problem geht es insbesondere darum, ob der Geist an den Körper gebunden ist oder nicht. Wir werden diese Fragen hier nicht diskutieren. Der interessierte Leser kann [Spe98, Spe97] zu Rate ziehen und ist eingeladen, sich im Rahmen des KI-Technologie-Studiums eine persönliche Meinung zu diesen Fragen zu bilden.

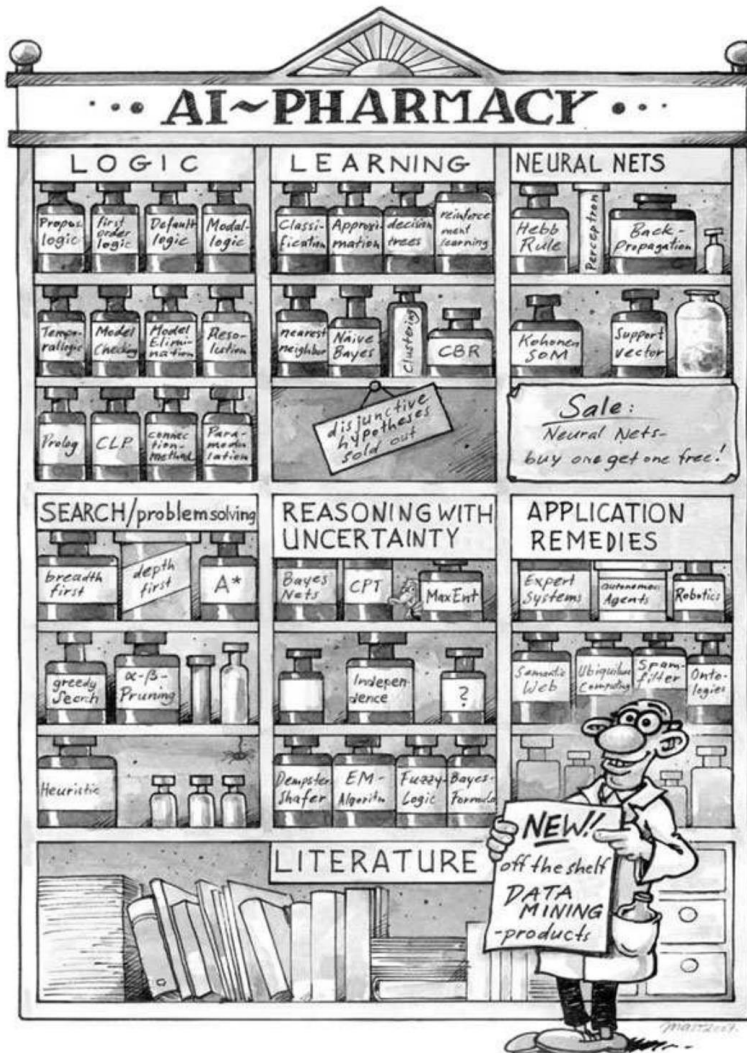


Abb. 1.2 Eine kleine Auswahl der von KI angebotenen Lösungen

1.1.2 Der Turing-Test und Chatterbots

Alan Turing machte sich als früher Pionier der KI einen Namen mit seiner Definition einer intelligenten Maschine, bei der die jeweilige Maschine den folgenden Test bestehen muss. Die Testperson Alice sitzt in einem verschlossenen Raum mit zwei Computerterminals. Ein Terminal ist mit einer Maschine verbunden, das andere mit einer nicht böswilligen Person, Bob. Alice kann in beide Terminals Fragen eingeben. Sie erhält die Aufgabe, nach fünf Minuten zu entscheiden, welches Terminal zum Automaten gehört. Die Maschine besteht den Test, wenn sie Alice in mindestens 30 % der Fälle austricksen kann [Tur50].

Während der Test philosophisch sehr interessant ist, ist er für die praktische KI, die sich mit der Problemlösung befasst, kein sehr relevanter Test. Die Gründe dafür sind ähnlich wie oben im Zusammenhang mit den Braitenberg-Fahrzeugen (siehe Übung 1.3 auf Seite 14).

Der KI-Pionier und Gesellschaftskritiker Joseph Weizenbaum entwickelte ein Programm namens *Eliza*, das die Fragen einer Testperson wie ein menschlicher Psychologe beantworten soll [Wei66]. Tatsächlich konnte er in vielen Fällen Erfolge vorweisen. Angeblich führte seine Sekretärin oft lange Diskussionen mit dem Programm. Heutzutage gibt es im Internet viele sogenannte *Chatterbots*, deren erste Reaktionen zum Teil recht beeindruckend sind. Nach einer gewissen Zeit wird jedoch ihr künstlicher Charakter deutlich. Einige dieser Programme sind tatsächlich lernfähig, während andere über außergewöhnliche Kenntnisse in verschiedenen Fächern verfügen, beispielsweise in Geographie oder Softwareentwicklung. Es gibt bereits kommerzielle Anwendungen für Chatterbots im Online-Kundensupport und möglicherweise weitere im Bereich E-Learning. Es ist denkbar, dass der Lernende und das E-Learning-System über einen Chatterbot kommunizieren. Der Leser möchte möglicherweise mehrere Chatterbots vergleichen und ihre Intelligenz in Übung 1.1 auf Seite 14 bewerten.

1.2 Die Geschichte der KI

KI greift auf viele wissenschaftliche Errungenschaften der Vergangenheit zurück, die hier nicht erwähnt werden, denn KI als eigenständige Wissenschaft existiert erst seit der Mitte des 20. Jahrhunderts. Tabelle 1.1 auf Seite 10 mit den wichtigsten KI-Meilensteinen und eine grafische Darstellung der Hauptbewegungen der KI in Abb. 1.3 auf Seite 6 ergänzen den folgenden Text.

1.2.1 Die ersten Anfänge

In den 1930er Jahren legten Kurt Gödel, Alonso Church und Alan Turing wichtige Grundlagen für die Logik und die theoretische Informatik. Von besonderem Interesse für die KI sind Gödels Theoreme. Der Vollständigkeitssatz besagt, dass die Prädikatenlogik erster Ordnung vollständig ist. Das bedeutet, dass jede wahre Aussage, die in der Prädikatenlogik formuliert werden kann, mit den Regeln einer formalen Analysis beweisbar ist. Auf dieser Grundlage könnten später automatische Theorembeweiser als Implementierung des Vollständigkeitssatzes entwickelt werden. Mit dem Unvollständigkeitssatz zeigte Gödel, dass es in Logiken höherer Ordnung wahre Aussagen gibt, die nicht beweisbar sind.² Damit deckte er schmerzhaft Grenzen formaler Systeme auf.

In diesen Zeitraum fällt auch Alan Turings Beweis der Unentscheidbarkeit des Halteproblems. Er zeigte, dass es kein Programm gibt, das entscheiden kann, ob ein bestimmtes beliebiges Programm (und seine entsprechende Eingabe) in einer Endlosschleife ausgeführt wird.

²Logiken höherer Ordnung sind Erweiterungen der Prädikatenlogik, in denen nicht nur Variablen, sondern auch Funktionssymbole oder Prädikate als Begriffe in einer Quantifizierung auftreten können. Tatsächlich hat Gödel nur gezeigt, dass jedes System, das auf Prädikatenlogik basiert und die Peano-Arithmetik formulieren kann, unvollständig ist.

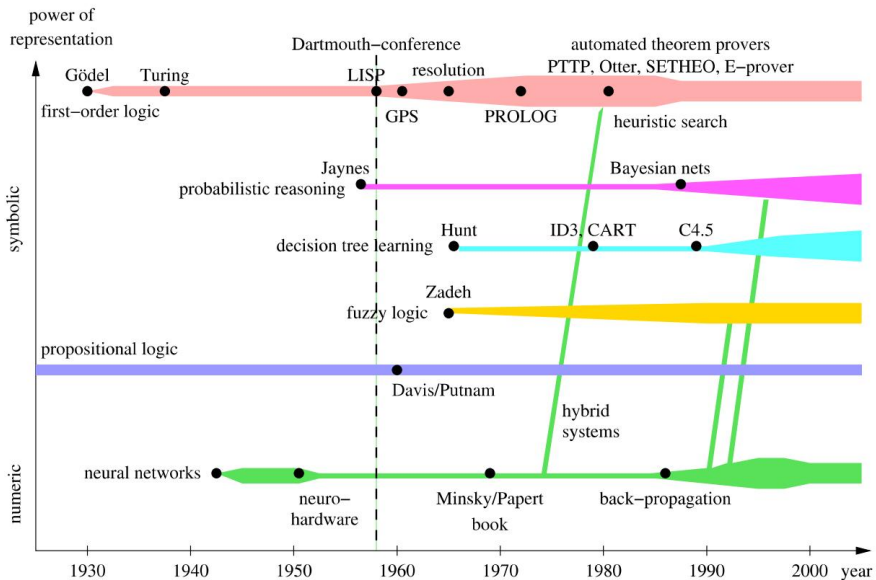


Abb. 1.3 Geschichte der verschiedenen KI-Bereiche. Die Breite der Balken zeigt die Prävalenz der Methodenverwendung an

Damit identifizierte Turing auch eine Grenze für intelligente Programme. Daraus folgt beispielsweise, dass es niemals ein universelles

Programmverifikationssystem geben wird.³ In den 1940er Jahren entwarfen McCulloch, Pitts und Hebb auf der Grundlage neurowissenschaftlicher Erkenntnisse die ersten mathematischen Modelle neuronaler Netze. Damals fehlte den Computern jedoch die nötige Leistung, um ein

1.2.2 Logik löst (fast) alle Probleme

KI als praktische Wissenschaft der Gedankenmechanisierung konnte natürlich erst beginnen, als es programmierbare Computer gab. Dies war in den 1950er Jahren der Fall. Newell und Simon führten mit Logic Theorist den ersten automatischen Theorembeweiser ein und zeigten damit auch, dass man mit Computern, die eigentlich nur mit Zahlen arbeiten, auch Symbole verarbeiten kann. Gleichzeitig führte McCarthy mit der Sprache LISP eine Programmiersprache ein, die speziell für die Verarbeitung symbolischer Strukturen geschaffen wurde. Beide Systeme wurden 1956 auf der historischen Dartmouth-Konferenz vorgestellt, die als Geburtstag der KI gilt.

In den USA entwickelte sich LISP zum wichtigsten Werkzeug zur Implementierung symbolverarbeitender KI-Systeme. Danach entwickelte sich die als Resolution bekannte logische Folgerungsregel zu einem vollständigen Kalkül der Prädikatenlogik.

³Diese Aussage gilt für „völlige Korrektheit“, was einen Nachweis der korrekten Ausführung sowie einen Nachweis der Beendigung für jede gültige Eingabe impliziert.

In den 1970er Jahren wurde die logische Programmiersprache PROLOG eingeführt. Europäisches Gegenstück zu LISP. PROLOG bietet den Vorteil, direkt zu ermöglichen Programmierung mit Horn-Klauseln, einer Teilmenge der Prädikatenlogik. Wie LISP, PROLOG verfügt über Datentypen zur komfortablen Verarbeitung von Listen.

Bis weit in die 1980er Jahre herrschte vor allem bei der KI ein Durchbruchgeist viele Logiker. Grund dafür waren die beeindruckenden Erfolge in Symbolverarbeitung. Mit dem Projekt „Computersysteme der fünften Generation“ in Japan und des ESPRIT-Programms in Europa wurden große Investitionen in den Bau getätigt Intelligente Computer.

Für kleine Probleme automatische Beweiser und andere Symbolverarbeitungssysteme hat manchmal sehr gut funktioniert. Die kombinatorische Explosion des Suchraums definierte jedoch ein sehr enges Fenster für diese Erfolge. Diese Phase der KI wurde in [RN10] als „Schau, Ma, keine Hände!“ beschrieben. Epoche.

Denn der wirtschaftliche Erfolg von KI-Systemen blieb hinter den Erwartungen zurück, die Finanzierung. Die Nachfrage nach logikbasierter KI-Forschung in den Vereinigten Staaten ging in den 1980er Jahren dramatisch zurück.

1.2.3 Der neue Konnektionismus

In dieser Phase der Ernüchterung treffen Informatiker, Physiker und Kognitive

Mit inzwischen ausreichend leistungsfähigen Computern konnten Wissenschaftler zeigen, dass mathematisch modellierte neuronale Netze lernfähig sind

Trainingsbeispiele, um Aufgaben auszuführen, die bisher eine aufwendige Programmierung erforderten.

Aufgrund der Fehlertoleranz solcher Systeme und ihrer Fähigkeit, Muster zu erkennen, waren vor allem bei der Mustererkennung erhebliche Erfolge möglich.

Gesichtserkennung in Fotos und Handschrifterkennung sind zwei Beispielanwendungen. Das System Nattalk konnte Sprache aus Beispieltextrn lernen [SR86].

Unter dem Namen *Konnektionismus* wurde eine neue Unterdisziplin der KI geboren.

Der Konnektionismus boomte und die Subventionen flossen. Aber bald ist auch hier die Machbarkeit gegeben. Grenzen wurden offensichtlich. Die neuronalen Netze könnten beeindruckende Fähigkeiten erlangen, aber es war meist nicht möglich, das erlernte Konzept in einfache Formeln oder Formeln zu fassen logische Regeln. Versuche, neuronale Netze mit logischen Regeln oder dem Wissen zu verbinden menschlicher Experten stieß auf große Schwierigkeiten. Darüber hinaus keine zufriedenstellende Lösung zur Strukturierung und Modularisierung der Netzwerke gefunden.

1.2.4 Argumentation unter Unsicherheit

KI als praktische, zielgerichtete Wissenschaft suchte nach einem Ausweg aus dieser Krise. Eins wollte die Fähigkeit der Logik, Wissen explizit darzustellen, mit der Stärke neuronaler Netze im Umgang mit Unsicherheit vereinen. Es wurden mehrere Alternativen vorgeschlagen.

Die vielversprechendste *probabilistische Argumentation* arbeitet mit bedingten Wahrscheinlichkeiten für Aussagenkalkülformeln. Seitdem wurden viele Diagnose- und Expertensysteme für Probleme des alltäglichen Denkens mithilfe von *Bayes'schen Netzen* entwickelt. Der Erfolg Bayes'scher Netzwerke beruht auf ihrer intuitiven Verständlichkeit

saubere Semantik der bedingten Wahrscheinlichkeit und aus der jahrhundertealten, mathematisch fundierten Wahrscheinlichkeitstheorie.

Die Schwächen der Logik, die nur mit zwei Wahrheitswerten arbeiten kann, können sein gelöst durch *Fuzzy-Logik*, die pragmatisch unendlich viele Werte zwischen Null und Eins einführt. Auch wenn seine theoretischen Grundlagen bis heute nicht völlig gesichert sind, insbesondere in der Regelungstechnik wird es erfolgreich eingesetzt.

Ein ganz anderer Weg führte zur erfolgreichen Synthese von Logik und neuronalen Netzen unter dem Namen *Hybridsysteme*. Hierzu wurden beispielsweise neuronale Netze eingesetzt Lernen Sie Heuristiken zur Reduzierung des riesigen kombinatorischen Suchraums bei der Beweisfindung [SE90].

Methoden des Entscheidungsbaumlernens aus Daten funktionieren auch mit Wahrscheinlichkeiten. Systeme wie CART, ID3 und C4.5 können schnell und automatisch sehr genaue Daten erstellen Entscheidungsbäume, die aussagenlogische Konzepte darstellen und dann verwendet werden können als Expertensysteme. Heute gehören sie zu den beliebtesten Techniken des maschinellen Lernens (Abschn. 8.4).

Seit etwa 1990 hat sich *Data Mining* als Teildisziplin der KI in diesem Bereich entwickelt der statistischen Datenanalyse zur Extraktion von Wissen aus großen Datenbanken. Daten Mining bringt keine neuen Techniken für die KI mit sich, sondern führt vielmehr dazu, dass große Datenbanken genutzt werden müssen, um explizites Wissen zu gewinnen. Eine Anwendung mit großem Markt Potenzial liegt in der Steuerung von Werbekampagnen großer Unternehmen auf Basis der Analyse von vielen Millionen Käufen ihrer Kunden. Typischerweise werden maschinelle Lerntechniken wie z als Entscheidungsbaumlernen kommen hier ins Spiel.

1.2.5 Verteilte, autonome und lernende Agenten

Verteilte künstliche Intelligenz (DAI) ist seit etwa 20 Jahren ein aktiver Forschungsbereich 1985. Eines seiner Ziele ist der Einsatz paralleler Computer zur Steigerung der Effizienz von Problemlösern. Es stellte sich jedoch heraus, dass dies auf den hohen Rechenaufwand zurückzuführen war Angesichts der Komplexität der meisten Probleme ist der Einsatz „intelligenter“ Systeme vorteilhafter als die Parallelisierung selbst.

Ein ganz anderer konzeptioneller Ansatz ergibt sich aus der Entwicklung autonomer Softwareagenten und Roboter, die wie menschliche Teams kooperieren sollen.

Wie bei den oben genannten Braitenberg-Fahrzeugen gibt es viele Fälle, in denen Ein einzelner Agent ist selbst mit unbegrenzten Ressourcen nicht in der Lage, ein Problem zu lösen. Erst die Zusammenarbeit vieler Akteure führt zum intelligenten Verhalten bzw. zu die Lösung eines Problems. Eine Ameisenkolonie oder eine Termitenkolonie kann sich errichten Gebäude von sehr hoher architektonischer Komplexität, obwohl es keine einzige Ameise gibt versteht, wie das Ganze zusammenpasst. Dies ähnelt der Situation von Bereitstellung von Brot für eine Großstadt wie New York [RN10]. Es gibt keine zentrale Planungsstelle für Brot, sondern Hunderte von Bäckern, die ihre jeweiligen Bedürfnisse kennen Bereiche der Stadt und backen Sie dort die entsprechende Menge Brot.

Der aktive Kompetenzerwerb durch Roboter ist ein spannendes Gebiet der aktuellen Forschung. Dort sind heute beispielsweise Roboter, die selbständig laufen lernen oder verschiedene Aufgaben ausführen motorische Fähigkeiten rund um den Fußball (Kap. 10). Kooperatives Lernen mehrerer Roboter zu Gemeinsam Probleme lösen steckt noch in den Kinderschuhen.

1.2.6 KI wird erwachsen

Die oben genannten Systeme, die KI heute anbietet, sind kein Universalrezept, sondern eine Werkstatt mit einer überschaubaren Anzahl an Werkzeugen für ganz unterschiedliche Aufgaben. Die meisten dieser Tools sind gut entwickelt und stehen als fertige Softwarebibliotheken zur Verfügung, oft mit komfortablen Benutzeroberflächen. Die Auswahl des richtigen Tools und dessen sinnvoller Einsatz im Einzelfall bleibt dem KI-Entwickler bzw. Knowledge Engineer überlassen. Wie jedes andere Handwerk erfordert auch dieses eine solide Ausbildung, die dieses Buch fördern

Wie kaum eine andere Wissenschaft ist KI interdisziplinär, denn sie stützt sich auf interessante Erkenntnisse aus so unterschiedlichen Bereichen wie Logik, Operations Research, Statistik, Regelungstechnik, Bildverarbeitung, Linguistik, Philosophie, Psychologie und Neurobiologie. Hinzu kommt das Themengebiet der jeweiligen Bewerbung.

Ein KI-Projekt erfolgreich zu entwickeln ist daher nicht immer so einfach, aber fast immer äußerst spannend.

1.3 Agenten

Obwohl der Begriff „*intelligente Agenten*“ für die KI nicht neu ist, hat er erst in den letzten Jahren unter anderem durch [RN10] an Bedeutung gewonnen. *Agent* bezeichnet ganz allgemein ein System, das Informationen verarbeitet und aus einer Eingabe eine Ausgabe erzeugt. Diese Wirkstoffe können auf viele verschiedene Arten klassifiziert werden.

In der klassischen Informatik werden vor allem *Softwareagenten* eingesetzt (Abb. 1.4 auf Seite 11). In diesem Fall besteht der Agent aus einem Programm, das aus Benutzereingaben ein Ergebnis berechnet.

In der Robotik hingegen werden *Hardware-Agenten* (auch Roboter genannt) eingesetzt, die zusätzlich über Sensoren und Aktoren verfügen (Abb. 1.5 auf Seite 11).

Mit den Sensoren kann der Agent seine Umgebung wahrnehmen. Mit den Aktoren führt es Aktionen aus und verändert seine Umgebung.

Hinsichtlich der Intelligenz des Agenten unterscheidet man zwischen *Reflexagenten*, die nur auf Eingaben reagieren, und *Agenten mit Gedächtnis*, die auch die Vergangenheit in ihre Entscheidungen einbeziehen können. Ein fahrender Roboter beispielsweise, der über seine Sensoren seine genaue Position (und die Zeit) kennt, hat als Reflexagent keine Möglichkeit, seine Geschwindigkeit zu bestimmen. Wenn es jedoch die Position in kurzen, diskreten Zeitschritten speichert, kann es so leicht seine Durchschnittsgeschwindigkeit im vorherigen Zeitintervall berechnen.

Wenn ein Reflexagent von einem deterministischen Programm gesteuert wird, stellt er eine Funktion der Menge aller Eingaben zur Menge aller Ausgaben dar. Ein Agent mit Gedächtnis hingegen ist im Allgemeinen keine Funktion. Warum? (Siehe Übung 1.5 auf Seite 14.) Reflexagenten sind in Fällen ausreichend, in denen das zu lösende Problem einen Markov-Entscheidungsprozess beinhaltet. Hierbei handelt es sich um einen Prozess, bei dem lediglich der aktuelle Zustand benötigt wird, um die optimale nächste Aktion zu bestimmen (siehe Kap. 10).

Ein mobiler Roboter, der sich in einem Gebäude von Raum 112 zu Raum 179 bewegen soll, führt andere Aktionen aus als ein Roboter, der sich zu Raum 105 bewegen soll. Mit anderen Worten, die Aktionen hängen vom Ziel ab. Solche Agenten werden als *zielorientiert bezeichnet*.

Tabelle 1.1 Meilensteine in der Entwicklung der KI von Gödel bis heute 1931 Der

Österreicher Kurt Gödel zeigt, dass in <i>der Prädikatenlogik erster Ordnung</i> alle wahren Aussagen ableitbar sind [Göd31a]. In der Logik höherer Ordnung hingegen gibt es wahre Aussagen, die nicht beweisbar sind [Göd31b]. (In [Göd31b] zeigte Gödel, dass die mit den Axiomen der Arithmetik erweiterte Prädikatenlogik unvollständig ist.)
1937 Alan Turing weist mit dem Halteproblem auf die Grenzen intelligenter Maschinen hin [Tur37].
1943 McCulloch und Pitts modellieren <i>neuronale Netze</i> und stellen die Verbindung zu propositionalen Netzen her Logik.
1950 Alan Turing definiert maschinelle Intelligenz mit dem <i>Turing-Test</i> und schreibt über lernende Maschinen und genetische Algorithmen [Tur50].
1951 Marvin Minsky entwickelt eine neuronale Netzwerkmaschine. Mit 3000 Vakuumröhren er simuliert 40 Neuronen.
1955 Arthur Samuel (IBM) entwickelt ein Lernschachprogramm, das besser spielt als sein Entwickler [Sam59].
1956 McCarthy organisiert eine Konferenz im Dartmouth College. Hier der Name <i>Artificial Intelligenz</i> wurde erstmals eingeführt. Newell und Simon von der Carnegie Mellon University (CMU) präsentieren <i>Logic Theorist</i> , das erste symbolverarbeitende Computerprogramm [NSS83].
1958 McCarthy erfindet am MIT (Massachusetts Institute of Technology) die Hochsprache <i>LISP</i> . Er schreibt Programme, die in der Lage sind, sich selbst zu modifizieren.
1959 Gelernter (IBM) entwickelt den Geometry Theorem Prover.
1961 Der General Problem Solver (GPS) von Newell und Simon imitiert menschliches Denken [NS61].
1963 McCarthy gründet das AI Lab an der Stanford University.
1965 Robinson erfindet den <i>Resolutionskalkül</i> für die Prädikatenlogik [Rob65] (Abschn. 3.5).
1966 Weizenbaums Programm Eliza führt Dialoge mit Menschen in natürlicher Sprache [Wei66] (Abschn. 1.1.2).
1969 zeigen Minsky und Papert in ihrem Buch <i>Perceptrons</i> , dass das Perzeptron ein sehr einfaches ist neuronales Netz, kann nur lineare Funktionen darstellen [MP69] (Abschn. 1.1.2).
1972 Der französische Wissenschaftler Alain Colmerauer erfindet die logische Programmiersprache <i>PROLOG</i> (Kap. 5). Der britische Arzt de Dombal entwickelt ein <i>Expertensystem</i> zur Diagnose akuter Bauchschmerzen [dDLS+72]. Dies bleibt in der damaligen Mainstream-KI-Community unbemerkt (Abschn. 7.3).
1976 Shortliffe und Buchanan entwickeln MYCIN, ein Expertensystem zur Diagnose von Infektionskrankheiten, das in der Lage ist, mit Unsicherheit umzugehen (Kap. 7).
1981 Japan startet mit großem Aufwand das „Projekt der fünften Generation“ mit dem Ziel Aufbau einer leistungsstarken PROLOG-Maschine.
1982 R1, das Expertensystem zur Konfiguration von Computern, spart Digital Equipment Corporation 40 Millionen Dollar pro Jahr [McD82].
1986 Renaissance neuronaler Netze unter anderem durch Rumelhart, Hinton und Sejnowski [RM86]. Das System Nettalk lernt, Texte laut vorzulesen [SR86] (Kap. 9).
1990 Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter bringen Wahrscheinlichkeitstheorie mit <i>Bayes'schen Netzwerken</i> in die KI (Abschn. 7.4). Multiagentensysteme erfreuen sich zunehmender Beliebtheit.
1992 Tesauros TD-Gammon-Programm demonstriert die Vorteile der Verstärkung Lernen.
1993 Weltweite <i>RoboCup</i> - Initiative zum Bau fußballspielender autonomer Roboter [Roba].

Tabelle 1.1 (Fortsetzung)

1995	Ausgehend von der statistischen Lerntheorie entwickelt Vapnik Support-Vektor-Maschinen, die heute eine große Bedeutung haben.
1997	IBMs Schachcomputer Deep Blue besiegt den Schachweltmeister Gary Kasparov. Erster internationaler RoboCup-Wettbewerb in Japan.
2003	Die Roboter im RoboCup zeigen eindrucksvoll, wozu KI und Robotik fähig sind erreichen.
2006	Servicerobotik wird zu einem wichtigen KI-Forschungsgebiet.
2010	Autonome Roboter beginnen, ihre Richtlinien zu erlernen.
2011	IBMs Programm „Watson“ zum Verständnis natürlicher Sprache und zur Beantwortung von Fragen besiegt zwei menschliche Champions in der US-Fernsehquizshow „Jeopardy!“. (Abschn. 1.4).

Abb. 1.4 Ein Softwareagent mit Benutzerinteraktion

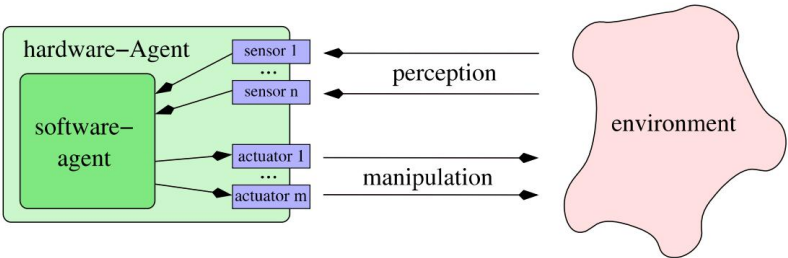
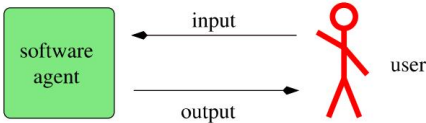


Abb. 1.5 Ein Hardware-Agent

Beispiel 1.1 Ein Spamfilter ist ein Agent, der eingehende E-Mails in gewünschte oder unerwünschte (Spam-)Kategorien einordnet und alle unerwünschten E-Mails löscht. Sein Ziel als zielorientierter Agent besteht darin, alle E-Mails in die richtige Kategorie einzuordnen. Bei dieser gar nicht so einfachen Aufgabe können dem Agenten gelegentlich Fehler unterlaufen. Da das Ziel darin besteht, alle E-Mails korrekt zu klassifizieren, wird versucht, so wenig Fehler wie möglich zu machen. Allerdings ist das nicht immer das, was der Benutzer im Sinn hat. Von 1.000 E-Mails macht Agent 1 nur 12 Fehler. Agent 2 hingegen macht bei denselben 1.000 E-Mails 38 Fehler. Ist es also schlechter als Agent 1? Die Fehler beider Agenten werden in der folgenden Tabelle, der sogenannten „Verwirrungsmatrix“, detaillierter dargestellt:

Agent 1:				Agent 2:			
		richtige Klasse				richtige Klasse	
		gesucht	gesuchter Spam			gesucht	gesuchter Spam
Der Spamfilter entscheidet	gesucht	189	1	Der Spamfilter entscheidet	gesucht	200	38
	Spam	11	799		762	3	0

Tatsächlich macht Agent 1 weniger Fehler als Agent 2, aber diese wenigen Fehler sind schwerwiegend, da der Benutzer 11 potenziell wichtige E-Mails verliert. Da es sich hierbei um zwei Arten von Fehlern unterschiedlicher Schwere handelt, sollte jeder Fehler mit dem entsprechenden Kostenfaktor gewichtet werden (siehe Abschn. 7.3.5 und Aufgabe 1.7 auf Seite 14).

Die Summe aller gewichteten Fehler ergibt die durch Fehlentscheidungen verursachten Gesamtkosten. Das Ziel eines *kostenbasierten Agenten* besteht darin, die Kosten von Fehlentscheidungen langfristig, also im Durchschnitt, zu minimieren. In Abschn. 7.3 lernen wir das Diagnosesystem LEXMED als Beispiel für einen kostenbasierten Agenten kennen.

Analog dazu besteht das Ziel eines nutzenbasierten Agenten darin, den Nutzen richtiger Entscheidungen langfristig, also im Durchschnitt, zu maximieren. Die Summe aller Entscheidungen gewichtet mit ihren jeweiligen Nutzenfaktoren ergibt den Gesamtnutzen.

Von besonderem Interesse in der KI sind *lernende Agenten*, die in der Lage sind, sich anhand von Trainingsbeispielen oder durch positives oder negatives Feedback so zu verändern, dass der durchschnittliche Nutzen ihrer Handlungen mit der Zeit wächst (siehe Kap. 8).

Wie in Abschn. 1.2.5 kommen zunehmend *verteilte Agenten* zum Einsatz, deren Intelligenz nicht in einem Agenten lokalisiert ist, sondern erst durch die Zusammenarbeit vieler Agenten sichtbar wird.

Das Design eines Agenten orientiert sich neben seinem Ziel stark an seiner *Umgebung* bzw. an seinem Bild der Umgebung, das stark von seiner Sensorik abhängt. Die Umgebung ist *beobachtbar*, wenn der Agent stets den vollständigen Zustand der Welt kennt. Ansonsten ist die Umgebung nur *teilweise beobachtbar*.

Wenn eine Aktion immer zum gleichen Ergebnis führt, dann ist die Umgebung *deterministisch*. Ansonsten ist es *nichtdeterministisch*. In einer *diskreten Umgebung* treten nur endlich viele Zustände und Aktionen auf, wohingegen eine *kontinuierliche Umgebung* unendlich viele Zustände oder Aktionen aufweist.

1.4 Wissensbasierte Systeme Ein Agent ist ein

Programm, das eine Abbildung von Wahrnehmungen auf Handlungen durchführt. Für einfache Agenten ist diese Sichtweise des Problems ausreichend. Bei komplexen Anwendungen, bei denen der Agent auf eine große Menge an Informationen zurückgreifen muss und eine schwierige Aufgabe erfüllen soll, kann die Programmierung des Agenten sehr kostspielig und unklar sein, wie vorzugehen ist. Hier bietet KI einen klaren Weg, der die Arbeit erheblich vereinfachen wird.

Zunächst trennen wir *das Wissen* vom System oder Programm, das das Wissen beispielsweise nutzt, um Schlussfolgerungen zu ziehen, Fragen zu beantworten oder einen Plan zu entwickeln. Dieses System wird als *Inferenzmechanismus bezeichnet*. Das Wissen wird in einer *Wissensdatenbank (KB) gespeichert*. Der Erwerb von Wissen in der Wissensbasis wird als *Knowledge Engineering* bezeichnet und basiert auf verschiedenen Wissensquellen wie menschlichen Experten, dem Wissensingenieur und Datenbanken. Aktiv lernende Systeme können Wissen auch durch aktive Erkundung der Welt erwerben (siehe Kap. 10). In Abb. 1.6 auf Seite 13 wird die allgemeine Architektur wissensbasierter Systeme dargestellt.

Der Übergang zu einer Trennung von Wissen und Schlussfolgerung hat mehrere entscheidende Vorteile. Durch die Trennung von Wissen und Inferenz können Inferenzsysteme weitgehend anwendungsunabhängig implementiert werden. Es ist zum Beispiel viel

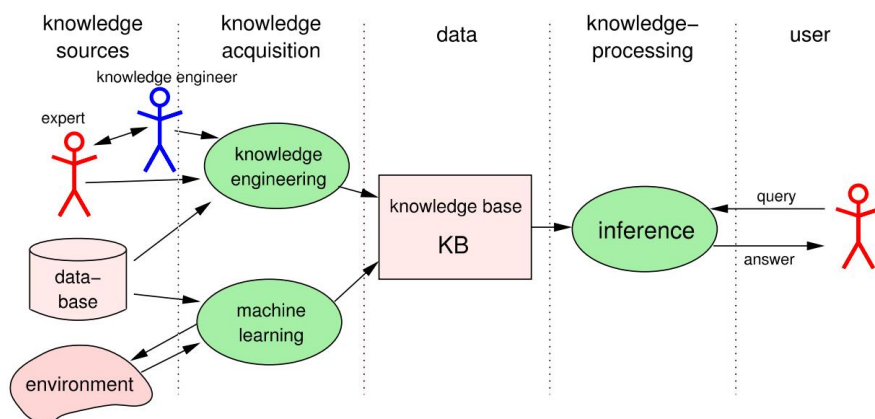


Abb. 1.6 Struktur eines klassischen Wissensverarbeitungssystems

Es ist einfacher, die Wissensbasis eines medizinischen Expertensystems zu ersetzen, als ein zu programmieren völlig neues System.

Durch die Entkopplung der Wissensbasis von der Schlussfolgerung kann Wissen deklarativ gespeichert werden. In der Wissensdatenbank gibt es nur eine Beschreibung davon Wissen, das unabhängig vom verwendeten Inferenzsystem ist. Ohne das klare Trennung, Wissen und Verarbeitung von Inferenzschritten würden miteinander verwoben, und jede Änderung des Wissens wäre sehr kostspielig.

Als praktische Schnittstelle zwischen Mensch und Maschine bietet sich die formale Sprache an zur Darstellung von Wissen in der Wissensdatenbank. In den folgenden Kapiteln werden wir eine ganze Reihe solcher Sprachen kennenlernen. Zuerst in Kap. 2 und 3

Es gibt *Aussagenlogik* und *Prädikatenlogik erster Ordnung* (PL1). Aber auch andere Malismen wie probabilistische Logik, Fuzzy-Logik oder Entscheidungsbäume werden vorgestellt.

Wir beginnen mit der Aussagenrechnung und den zugehörigen Folgerungssystemen. Aufbauend auf Daraufhin stellen wir die Prädikatenlogik vor, eine mächtige Sprache, die für Maschinen zugänglich und in der KI sehr wichtig ist.

Als Beispiel für ein groß angelegtes wissensbasiertes System möchten wir auf Folgendes verweisen Software-Agent „Watson“. Entwickelt bei IBM zusammen mit mehreren Universitäten, Watson ist ein Frage-Antwort-Programm, das mit natürlichen Hinweisen gefüttert werden kann Sprache. Es basiert auf einer Wissensbasis, die vier Terabyte Festplattenspeicher umfasst, darunter auch den Volltext von Wikipedia [FNA+09]. Watson wurde innerhalb entwickelt Das DeepQA-Projekt von IBM, das in [Dee11] wie folgt charakterisiert wird:

Das DeepQA-Projekt bei IBM stellt eine große Herausforderung in der Informatik dar, die darauf abzielt veranschaulichen, wie die umfassende und zunehmende Zugänglichkeit von Inhalten in natürlicher Sprache und die Integration und Weiterentwicklung der Verarbeitung natürlicher Sprache, des Informationsabrufs, des maschinellen Lernens, der Wissensrepräsentation und des Denkens sowie massiv paralleler Berechnungen die automatische Frage-Antwort-Technologie im offenen Bereich vorantreiben können. Punctes konkurriert klar und konsequent mit der besten menschlichen Leistung.

In der US-Fernsehquizshow „Jeopardy!“ besiegte Watson im Februar 2011 die beiden menschlichen Champions Brad Rutter und Ken Jennings in einem Zwei-Spiele-Spiel. Kombiniertes Punktespiel und gewann den Preis von einer Million Dollar. Einer von Watsons Kollegen

Besondere Stärken waren die sehr schnelle Reaktion auf die Fragen, was dazu führte, dass Watson den Summer (mittels eines Magnetventils) oft schneller als seine menschlichen Konkurrenten betätigte und dann die erste Antwort auf die Frage geben konnte.

Die hohe Leistung und die kurzen Reaktionszeiten von Watson sind auf eine Implementierung auf 90 IBM Power 750-Servern zurückzuführen, die jeweils 32 Prozessoren enthalten, was zu 2880 parallelen Prozessoren führt.

1.5 Übungen

Übung 1.1 Testen Sie einige der im Internet verfügbaren Chatterbots. Beginnen Sie zum Beispiel mit www.hs-weingarten.de/~ertel/aibook in der Linksammlung unter Turingtest/Chatterbots, oder unter www.simonlaven.com oder www.alicebot.org. Schreiben Sie eine Ausgangsfrage auf und messen Sie die Zeit, die für jedes der verschiedenen Programme benötigt wird, bis Sie sicher wissen, dass es sich nicht um einen Menschen handelt.

Übung 1.2 Unter www.pandorabots.com finden Sie einen Server, auf dem Sie ganz einfach einen Chatterbot mit der Auszeichnungssprache AIML erstellen können. Entwickeln Sie je nach Interesse einen einfachen oder komplexen Chatterbot oder ändern Sie einen bestehenden.

Aufgabe 1.3 Begründen Sie die Ungeeignetheit des Turing-Tests als Definition von „*künstlicher Intelligenz*“ in der praktischen KI.

Übung 1.4 Viele bekannte Inferenzprozesse, Lernprozesse usw. sind NP-vollständig oder sogar unentscheidbar. Was bedeutet das für KI?

Aufgabe 1.5 (a)

Warum ist ein deterministischer Agent mit Gedächtnis im mathematischen Sinne keine Funktion von der Menge aller Eingaben zur Menge aller Ausgaben? (b) Wie kann man den Agenten mit dem Gedächtnis verändern oder ihn so modellieren, dass er wird?

Äquivalent zu einer Funktion, verliert aber nicht ihr Gedächtnis?

Übung 1.6 Es gebe einen Agenten mit Gedächtnis, der sich innerhalb einer Ebene bewegen kann. Von seinen Sensoren erhält es in regelmäßigen Takten t seine genaue Position (x, y) in kartesischen Koordinaten. (a) Geben Sie

eine Formel an, mit der der Agent seine Geschwindigkeit aus der Strömung berechnen kann
Zeit t und die vorherige Messung von $t \rightarrow t$.

(b) Wie muss der Agent verändert werden, damit er auch seine Beschleunigung berechnen kann? Geben Sie auch hier eine Formel an.

Übung 1.7

(a) Bestimmen Sie für beide Agenten in Beispiel 1.1 auf Seite 11 die durch die Fehler verursachten Kosten und vergleichen Sie die Ergebnisse. Gehen Sie hier davon aus, dass das manuelle Löschen einer Spam-E-Mail einen Cent kostet und das Wiederherstellen einer gelöschten E-Mail oder der Verlust einer E-Mail einen Dollar kostet.

(b) Bestimmen Sie für beide Agenten den Gewinn, der durch korrekte Klassifizierungen entsteht, und vergleichen Sie die Ergebnisse. Gehen Sie davon aus, dass für jede gewünschte erkannte E-Mail ein Gewinn von einem Dollar anfällt und für jede korrekt gelöschte Spam-E-Mail ein Gewinn von einem Cent.

Aussagenlogik

2

In der Aussagenlogik werden, wie der Name schon sagt, Aussagen durch logische Operatoren verknüpft. Die Aussage „*die Straße ist nass*“ ist eine Aussage, ebenso wie „*es regnet*“. Diese beiden Sätze können zu dem neuen Satz verbunden werden

Wenn es regnet, ist die Straße nass.

Formeller geschrieben

es regnet $\dot{\vee}$ *die Straße ist nass.*

Diese Notation hat den Vorteil, dass die Elementarsätze in unveränderter Form wieder auftauchen. Damit wir präzise mit Aussagenlogik arbeiten können, beginnen wir mit einer Definition der Menge aller Aussagenlogikformeln.

2.1 Syntax

Definition 2.1 Sei $Op = \{\neg, \dot{\vee}, \dot{\wedge}, \dot{\rightarrow}, \dot{\leftrightarrow}, (,)\}$ die Menge der logischen Operatoren und $\dot{\gamma}$ eine Menge der Symbole. Die Mengen Op , $\dot{\gamma}$ und $\{t, f\}$ sind paarweise disjunkt. $\dot{\gamma}$ heißt *Signatur* und seine Elemente sind die *Satzvariablen*. Die Menge der aussagenlogischen Formeln ist nun rekursiv definiert:

- t und f sind (atomare) Formeln.
- Alle Satzvariablen, also alle Elemente aus $\dot{\gamma}$, sind (atomare) Formeln las.
- Wenn A und B Formeln sind, dann sind $\neg A$, (A) , $A \dot{\vee} B$, $A \dot{\wedge} B$, $A \dot{\rightarrow} B$, $A \dot{\leftrightarrow} B$ ebenfalls Formeln.

Diese elegante rekursive Definition der Menge aller Formeln ermöglicht uns die Generierung unendlich viele Formeln. Zum Beispiel gegeben $\dot{\gamma} = \{A, B, C\}$,

$A \dot{\vee} B$, $A \dot{\vee} B \dot{\vee} C$, $A \dot{\vee} A \dot{\vee} A$, $C \dot{\vee} B \dot{\vee} A$, $(\neg A \dot{\vee} B) \dot{\vee} (\neg C \dot{\vee} A)$

sind Formeln. $((A)) \dot{\vee} B$ ist ebenfalls eine syntaktisch korrekte Formel.

Definition 2.2 Wir lesen die Symbole und Operatoren folgendermaßen:

t :	„wahr“	
f :	„falsch“	
$\neg A$:	„nicht A“	(Negation)
$A \wedge B$:	„A und B“	(Verbindung)
$A \vee B$:	„A oder B“	(Disjunktion)
$A \rightarrow B$:	„Wenn A, dann B“	(Implikation (auch <i>materielle Implikation</i> genannt))
$A \leftrightarrow B$:	„A genau dann, wenn B“	(Äquivalenz)

Die so definierten Formeln sind bisher rein syntaktische Konstruktionen ohne Bedeutung. Uns fehlt noch die Semantik.

2.2 Semantik

In der Aussagenlogik gibt es zwei Wahrheitswerte: t für „wahr“ und f für „falsch“. Wir Beginnen Sie mit einem Beispiel und fragen Sie sich, ob die Formel $A \wedge B$ wahr ist. Der Die Antwort lautet: Es kommt darauf an, ob die Variablen A und B wahr sind. Wenn zum Beispiel A steht für „Heute regnet es“ und B für „Heute ist es kalt“ und beides ist wahr, dann ist $A \wedge B$ wahr. Wenn B jedoch „Heute ist es heiß“ darstellt (und das ist falsch), dann $A \wedge B$ ist falsch.

Wir müssen dem Propo offensichtlich Wahrheitswerte zuweisen, die den Zustand der Welt widerspiegeln Positionsvariablen. Deshalb definieren wir

Definition 2.3 Eine Abbildung $I : \mathcal{F} \rightarrow \{w, f\}$, die einen Wahrheitswert zuordnet Jede Aussagevariable wird *Interpretation* genannt.

Da jede Aussagevariable zwei Wahrheitswerte annehmen kann, hat jede aussagenlogische Formel mit n verschiedenen Variablen 2^n verschiedene Interpretationen. Wir Definieren Sie die Wahrheitswerte für die Grundoperationen, indem Sie alle möglichen Interpretationen in einer *Wahrheitstabelle* darstellen (siehe Tabelle 2.1 auf Seite 17).

Die leere Formel gilt für alle Interpretationen. Um die Wahrheit herauszufinden Für komplexe Formeln müssen wir auch die Reihenfolge der Operationen für logische Werte definieren Betreiber. Wenn Ausdrücke in Klammern gesetzt sind, wird der Term in den Klammern ausgewertet

Tabelle 2.1 Definition derlogische Operatoren durch Wahrheit
Tisch

AB (A) $\neg A$ $A \dot{\vee} B$ $A \dot{\wedge} B$ $A \dot{\supset} B$ $A \dot{\equiv} B$				
ttt ft		T	T	T
t ft ff		T	F	F
ft ft	F	T	T	F
fff t	F	F	T	T

Erste. Bei Formeln ohne Klammern sind die Prioritäten wie folgt geordnet, beginnend mit der stärksten Bindung: $\neg, \dot{\vee}, \dot{\wedge}, \dot{\supset}, \dot{\equiv}$.

Klare Unterscheidung zwischen der Äquivalenz von Formeln und der syntaktischen Äquivalenz. Alence, wir definieren

Definition 2.4 Zwei Formeln F und G heißen semantisch äquivalent, wenn sie nehmen für alle Interpretationen den gleichen Wahrheitswert an. Wir schreiben $F \dot{\equiv} G$.

Semantische Äquivalenz dient vor allem dazu, die Metasprache nutzen zu können ist, natürliche Sprache, um über die Objektsprache zu sprechen, nämlich Logik. Die Aussage „ $A \dot{\equiv} B$ “ vermittelt, dass die beiden Formeln A und B semantisch äquivalent sind. Der Die Aussage „ $A \dot{\equiv} B$ “ hingegen ist ein syntaktisches Objekt der formalen Sprache der Aussagenlogik.

Je nachdem, in wie vielen Interpretationen eine Formel wahr ist, können wir einteilen Formeln in die folgenden Klassen:

Definition 2.5 Eine Formel heißt

- *Erfüllbar*, wenn es für mindestens eine Interpretation wahr ist.
- *Logisch gültig* oder einfach *gültig*, wenn es für alle Interpretationen wahr ist. Zwar werden Mulas auch *Tautologien* genannt.
- *Unerfüllbar*, wenn es für keine Interpretation wahr ist.

Jede Interpretation, die einer Formel genügt, wird als *Modell* der Formel bezeichnet.

Offensichtlich ist die Negation jeder allgemeingültigen Formel unerfüllbar. Die Negation einer erfüllbaren, aber nicht allgemein gültigen Formel F ist erfüllbar.

Wir sind nun in der Lage, Wahrheitstabellen für komplexe Formeln zu erstellen, um deren Wahrheitsgehalt zu ermitteln Wahrheitswerte. Wir setzen dies mithilfe von Formeläquivalenzen sofort in die Tat um die in der Praxis wichtig sind.

Satz 2.1 Die Operationen $\neg$, $\wedge$ sind kommutativ und assoziativ, und die Folgende Äquivalenzen gelten grundsätzlich:

$\neg A \wedge B \wedge A \wedge B$	(Implikation)
$A \wedge B \wedge \neg B \wedge \neg A$	(Kontraposition)
$(A \wedge B) \wedge (B \wedge A) \wedge (A \wedge B) \wedge \neg(A \wedge B) \wedge \neg A \wedge \neg B$	(Gleichwertigkeit)
$\neg(A \wedge B) \wedge \neg A \wedge \neg B$	(De Morgans Gesetz)
$A \wedge (B \wedge C) \wedge (A \wedge B) \wedge (A \wedge C)$	(Distributivgesetz)
$A \wedge (B \wedge C) \wedge (A \wedge B) \wedge (A \wedge C)$	
$A \wedge \neg A \wedge w$	(Tautologie)
$A \wedge \neg A \wedge f$	(Widerspruch)
$A \wedge f \wedge A$	
$A \wedge w \wedge w$	
$A \wedge f \wedge f$	
$A \wedge w \wedge A$	

Beweis Um die erste Äquivalenz zu zeigen, berechnen wir die Wahrheitstabelle für $\neg A \wedge B$ und $A \wedge B$ und stellen Sie sicher, dass die Wahrheitswerte für beide Formeln für alle Interpretationen gleich sind. Die Formeln sind also äquivalent und somit auch alle Werte der letzten Spalte sind „t“s.

$AB \neg A \neg A \wedge B A \wedge B (\neg A \wedge B) \wedge (A \wedge B)$		
t t f t t f f t t	T	T
	F	T
	T	T
	T	T

Die Beweise für die anderen Äquivalenzen sind ähnlich und werden als Übungen empfohlen für den Leser (Übung 2.2 auf Seite 29).

2.3 Beweissysteme

Bei der KI geht es uns darum, vorhandenes Wissen zu übernehmen und daraus Neues abzuleiten Wissen oder die Beantwortung von Fragen. In der Aussagenlogik bedeutet das, das zu zeigen eine *Wissensbasis* KB – also eine (möglicherweise umfangreiche) Aussagenlogikformel – Es folgt eine Formel Q1. Daher definieren wir zunächst den Begriff „Folge“.

¹Hier steht Q für Abfrage.

Definition 2.6 Eine Formel KB *beinhaltet* eine Formel Q (oder Q *folgt* aus KB), wenn jedes Modell von KB auch ein Modell von Q ist. Wir schreiben $KB \models Q$.

Mit anderen Worten: In jeder Interpretation, in der KB wahr ist, ist auch Q wahr. Kurz gesagt: Wann immer KB wahr ist, ist auch Q wahr. Da für den Begriff der Folgerung Interpretationen von Variablen herangezogen werden, handelt es sich um einen semantischen Begriff.

Jede Formel, die nicht gültig ist, wählt sozusagen eine Teilmenge der Menge aller Interpretationen als ihr Modell. Tautologien wie $A \vee \neg A$ beispielsweise schränken die Zahl der zufriedenstellenden Interpretationen nicht ein, da ihr Satz leer ist. Die leere Formel gilt daher in allen Interpretationen. Für jede Tautologie T gilt $\models T$. Intuitiv bedeutet dies, dass Tautologien immer wahr sind, ohne Einschränkung der Interpretationen durch eine Formel. Kurz gesagt schreiben wir $\models T$. Nun zeigen wir einen wichtigen Zusammenhang zwischen dem semantischen Konzept der Folgerung und der syntaktischen Implikation.

Satz 2.2 (Deduktionstheorem)

$A \models B$ genau dann, wenn $\vdash A \rightarrow B$.

Beweis Beachten Sie die Wahrheitstabelle zur Implikation:

A	B	$A \rightarrow B$
t	t	t
t	f	f
f	t	t
f	f	t

Eine willkürliche Implikation $A \rightarrow B$ ist offensichtlich immer wahr, außer bei der Interpretation $A = \text{t}, B = \text{f}$. Nehmen Sie an, dass $A \models B$ hält. Das bedeutet, dass für jede Interpretation, die A wahr macht, auch B wahr ist. Die kritische zweite Zeile der Wahrheitstabelle gilt in diesem Fall nicht einmal. Daher ist $A \rightarrow B$ wahr, was bedeutet, dass $A \rightarrow B$ eine Tautologie ist. Damit ist eine Richtung der Aussage aufgezeigt.

Nehmen Sie nun an, dass $A \rightarrow B$ gilt. Somit ist die kritische zweite Zeile der Wahrheitstabelle auch ausgespart. Jedes Modell von A ist dann auch ein Modell von B . Dann ist $A \models B$ halt.

Wenn wir zeigen wollen, dass $KB \models Q$ mit sich bringt, können wir auch mit der Wahrheitstabellenmethode zeigen, dass $KB \rightarrow Q$ eine Tautologie ist. Damit haben wir unser erstes *Beweissystem* für Aussagenlogik, das leicht zu automatisieren ist. Der Nachteil dieser Methode ist die im schlimmsten Fall sehr lange Rechenzeit. Konkret muss im schlimmsten Fall mit n Aussagevariablen für alle 2^n Interpretationen der Variablen die Formel $KB \rightarrow Q$ ausgewertet werden. Die Rechenzeit wächst daher exponentiell.

mit der Anzahl der Variablen. Daher ist dieses Verfahren zumindest im schlimmsten Fall für große Variablenzahlen unbrauchbar.

Wenn eine Formel KB eine Formel Q mit sich bringt, dann ist nach dem Deduktionssatz $KB \dot{\vee} Q$ eine Tautologie. Daher ist die Negation $\neg(KB \dot{\vee} Q)$ unerfüllbar. Wir haben

$$\neg(KB \dot{\vee} Q) \dot{\vee} \neg(\neg KB \dot{\vee} Q) \dot{\vee} KB \dot{\vee} \neg Q.$$

Daher ist auch $KB \dot{\vee} \neg Q$ erfüllbar. Wir formulieren diese einfache, aber wichtige Konsequenz des Deduktionssatzes als Satz.

Satz 2.3 (Beweis durch Widerspruch) $KB \mid Q$ genau dann, wenn $KB \dot{\vee} \neg Q$ unerfüllbar ist.

Um zu zeigen, dass die Abfrage Q aus der Wissensbasis KB folgt, können wir auch die negierte Abfrage $\neg Q$ zur Wissensbasis hinzufügen und einen Widerspruch ableiten. Aufgrund der Äquivalenz $A \dot{\vee} \neg A \dot{\vee} f$ aus Satz 2.1 auf Seite 18 wissen wir, dass ein Widerspruch unerfüllbar ist. Damit ist Q bewiesen. Dieses in der Mathematik häufig verwendete Verfahren wird auch in verschiedenen automatischen Beweiskalkülen wie der Auflösungsrechnung und bei der Abarbeitung von PROLOG-Programmen eingesetzt.

Eine Möglichkeit, zu vermeiden, dass alle Interpretationen mit der Wahrheitstabellenmethode überprüft werden müssen, ist die syntaktische Manipulation der Formeln KB und Q durch Anwendung von Inferenzregeln mit dem Ziel, sie stark zu vereinfachen, sodass wir am Ende sofort erkennen können, dass $KB \mid F$. Wir nennen diesen syntaktischen Prozess *Ableitung* und schreiben $KB \dot{\vee} Q$. Solche syntaktischen Beweissysteme werden *Kalküle* genannt. Um sicherzustellen, dass ein Kalkül keine Fehler erzeugt, definieren wir zwei grundlegende Eigenschaften von Ka

Definition 2.7 Ein Kalkül heißt *Klang*, wenn jeder abgeleitete Satz semantisch folgt. Das heißt, wenn für die Formeln KB und Q das gilt

$$\text{wenn } KB \dot{\vee} Q, \text{ dann } KB \mid Q.$$

Ein Kalkül heißt *vollständig*, wenn alle semantischen Konsequenzen abgeleitet werden können. Das heißt, für die Formeln KB und Q gilt:

$$\text{wenn } KB \mid Q \text{ dann } KB \dot{\vee} Q.$$

Die Solidität eines Kalküls stellt sicher, dass alle abgeleiteten Formeln tatsächlich semantische Konsequenzen der Wissensbasis sind. Das Kalkül erzeugt keine „falschen Konsequenzen“. Die Vollständigkeit eines Kalküls hingegen stellt sicher, dass der Kalkül nichts übersieht. Ein vollständiger Kalkül findet immer dann einen Beweis, wenn die zu beweisende Formel aus der Wissensbasis folgt. Wenn eine Rechnung korrekt ist

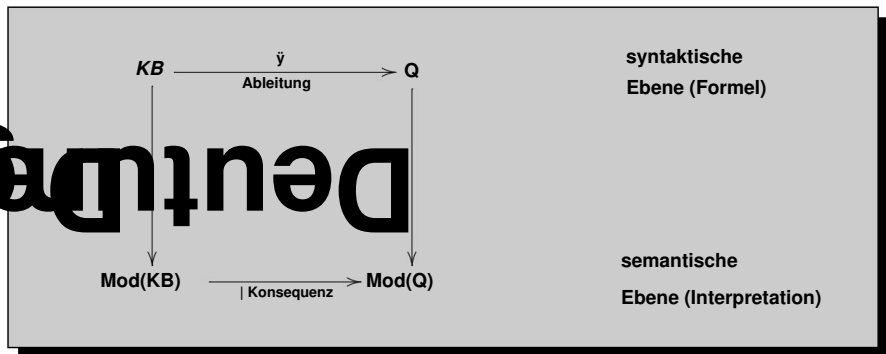


Abb. 2.1 Syntaktische Ableitung und semantische Folgerung. $\text{Mod}(X)$ stellt die Menge der Modelle einer Formel X dar

und vollständig, dann sind syntaktische Ableitung und semantische Folgerung zwei äquivalente Beziehungen (siehe Abb. 2.1).

Um automatische Proofsysteme möglichst einfach zu halten, werden diese üblicherweise hergestellt Formeln in konjunktiver Normalform bearbeiten.

Definition 2.8 Eine Formel liegt **genau dann in der konjunktiven Normalform (KNF)** vor, wenn sie aus einer **Konjunktion** besteht

$$K_1 \wedge K_2 \wedge \dots \wedge K_m$$

von Klauseln. Eine Klausel K_i besteht aus einer **Disjunktion**

$$(L_{i1} \vee L_{i2} \vee \dots \vee L_{ini})$$

von Literalen. Schließlich ist ein **Literal** eine Variable (positives Literal) oder eine negierte Variable (negatives Literal).

Die Formel $(A \wedge B \wedge \neg C) \wedge (A \wedge B) \wedge (\neg B \wedge \neg C)$ liegt in konjunktiver Normalform vor. Die konjunktive Normalform stellt keine Einschränkung für die Formelmenge dar, weil:

Satz 2.4 Jede aussagenlogische Formel lässt sich in eine äquivalente konjunktive Normalform umwandeln.

Beispiel 2.1 Wir bringen $A \vee B \vee C \vee D$ in die konjunktive Normalform, indem wir die Äquivalenzen aus Satz 2.1 auf Seite 18 verwenden:

$$\begin{aligned} A \vee B \vee C \vee D &\equiv \neg(\neg(A \vee B \vee C \vee D)) \\ &\equiv \neg(\neg A \wedge \neg B \wedge \neg C \wedge \neg D) \quad (\text{de Morgan}) \\ &\equiv (\neg \neg A \vee \neg \neg B \vee \neg \neg C \vee \neg \neg D) \quad (\text{Distributivgesetz}) \\ &\equiv (A \vee B \vee C \vee D) \quad (\text{Assoziativgesetz}) \end{aligned}$$

Jetzt fehlt uns nur noch ein Kalkül zum syntaktischen Beweis aussagenlogischer Formeln. Wir beginnen mit dem *Modus Ponens*, einer einfachen, intuitiven Schlussregel, die aus der Gültigkeit von A und $A \vee B$ die Ableitung von B ermöglicht. Wir schreiben dies formal als

$$\frac{A, A \vee B}{B}.$$

Diese Notation bedeutet, dass wir die Formel(n) unterhalb der Zeile aus den durch Kommas getrennten Formeln oberhalb der Zeile ableiten können. Modus ponens ist in der Regel für sich genommen zwar gesund, aber nicht vollständig. Wenn wir zusätzliche Regeln hinzufügen, können wir eine vollständige Rechnung erstellen, auf die wir hier jedoch nicht eingehen möchten. Stattdessen werden wir die *Auflösungsregel* untersuchen

$$\frac{A \vee B, \neg B \vee C}{A \vee C} \quad (2.1)$$

als Alternative. Der abgeleitete Satz heißt *resolvent*. Durch eine einfache Transformation erhalten wir die äquivalente Form

$$\frac{A \vee B, B \vee C}{A \vee C}.$$

Wenn wir A auf f setzen, sehen wir, dass die Auflösungsregel eine Verallgemeinerung des Modus Ponens ist. Die Auflösungsregel ist gleichermaßen anwendbar, wenn C fehlt oder wenn A und C fehlen. Im letzteren Fall lässt sich der Leersatz aus dem Widerspruch $B \vee \neg B$ ableiten (Übung 2.7 auf Seite 30).

2.4 Auflösung

Wir verallgemeinern nun die Auflösungsregel noch einmal, indem wir Klauseln mit einer beliebigen Anzahl von Literalen zulassen. Bei den Literalen $A_1, \dots, A_m, B, C_1, \dots, C_n$ lautet die *allgemeine Auflösungsregel*

$$\frac{(A_1 \vee \dots \vee A_m \vee B), (\neg B \vee C_1 \vee \dots \vee C_n)}{(A_1 \vee \dots \vee A_m \vee C_1 \vee \dots \vee C_n)} \quad (2.2)$$

Wir nennen die Literale B und $\neg B$ komplementär. Die Auflösungsregel löscht ein Paar komplementärer Literale aus den beiden Klauseln und kombiniert die restlichen Literale zu einer neuen Klausel.

Um zu beweisen, dass aus einer Wissensbasis KB eine Abfrage Q folgt, führen wir einen Widerspruchsbeweis durch. Nach Satz 2.3 auf Seite 20 müssen wir zeigen, dass aus $KB \vee \neg Q$ ein Widerspruch abgeleitet werden kann. In Formeln in konjunktiver Normalform tritt ein Widerspruch in Form zweier Sätze (A) und $(\neg A)$ auf, die auf den Leersatz als deren Auflösung führen. Der folgende Satz stellt uns sicher, dass dieser Prozess wirklich wie gewünscht funktioniert.

Damit die Rechnung vollständig ist, benötigen wir eine kleine Ergänzung, wie das folgende Beispiel zeigt. Als Wissensbasis sei die Formel $(A \vee A)$ gegeben. Um mit der Auflösungsregel zu zeigen, dass wir von dort $(A \vee A)$ ableiten können, müssen wir zeigen, dass die leere Klausel aus $(A \vee A) \vee (\neg A \vee \neg A)$ abgeleitet werden kann. Mit der Auflösungsregel allein ist dies unmöglich. Mit der Faktorisierung, die das Löschen von Kopien von Literalen aus Klauseln ermöglicht, wird dieses Problem beseitigt. Im Beispiel führt eine doppelte Anwendung der Faktorisierung zu $(A) \vee (\neg A)$ und ein Auflösungsschritt zur leeren Klausel.

Satz 2.5 Der Auflösungskalkül für den Beweis der Unerfüllbarkeit von Mulas in konjunktiver Normalform ist solide und vollständig.

Denn es ist die Aufgabe des Auflösungskalküls, daraus einen Widerspruch abzuleiten $KB \vee \neg Q$, es ist sehr wichtig, dass die Wissensbasis KB konsistent ist :

Definition 2.9 Eine Formel KB heißt *konsistent*, wenn sich daraus kein Widerspruch ableiten lässt, also eine Formel der Form $\bigvee \neg \bigvee$.

Ansonsten lässt sich aus KB alles ableiten (siehe Übung 2.8 auf Seite 30). Das gilt nicht nur für die Auflösung, sondern auch für viele andere Kalküle.

Unter den Berechnungen für den automatisierten Abzug spielt die Auflösung eine herausragende Rolle. Daher möchten wir noch etwas intensiver damit zusammenarbeiten. Im Gegensatz zu anderen Kalkülen hat die Auflösung nur zwei Inferenzregeln und funktioniert mit Formeln in konjunktiver Normalform. Dies vereinfacht die Umsetzung. Ein weiterer Vorteil im Vergleich zu vielen Berechnungen liegt in der Reduzierung der Möglichkeiten zur Anwendung von Inferenzregeln in jedem Schritt des Beweises, wodurch der Suchraum reduziert und die Rechenzeit verkürzt wird.

Als Beispiel beginnen wir mit einem einfachen Logikrätsel, das die wichtigen Schritte ermöglicht eines vorzulegenden Lösungsnachweises.

Beispiel 2.2 Logikrätsel Nummer 7 mit dem Titel „Eine bezaubernde englische Familie“ aus dem deutschen Buch [Ber89] lautet (übersetzt ins Englische):

Obwohl ich sieben Jahre lang mit großem Erfolg Englisch gelernt habe, muss ich zugeben, dass ich völlig perplex bin, wenn ich Engländer Englisch sprechen höre. Kürzlich nahm ich, von edlen Gefühlen bewegt, drei Anhalter mit, einen Vater, eine Mutter und eine Tochter, von denen ich schnell merkte, dass sie Engländer waren und nur Englisch sprachen. Bei jedem der folgenden Sätze schwankte ich zwischen zwei möglichen Interpretationen. Sie sagten mir Folgendes (die zweite mögliche Bedeutung steht in Klammern): Der Vater: „Wir fahren nach Spanien (wir kommen aus Newcastle).“

Die Mutter: „Wir fahren nicht nach Spanien und kommen aus Newcastle (wir haben in Paris Halt gemacht und fahren nicht nach Spanien).“ Die Tochter: „Wir kommen nicht aus Newcastle (wir haben in Paris Halt gemacht).“

Was ist mit dieser bezaubernden englischen Familie?

Um ein solches Problem zu lösen, gehen wir in drei Schritten vor: Formalisierung, Transformation in Normalform und Beweis. In vielen Fällen ist die Formalisierung der mit Abstand schwierigste Schritt, da leicht Fehler gemacht oder kleine Details vergessen werden können. (Daher ist die praktische Übung sehr wichtig. Siehe Übungen 2.9–2.11.)

Hier verwenden wir die Variablen S für „Wir fahren nach Spanien“, N für „Wir kommen aus Newcastle“ und P für „Wir haben in Paris angehalten“ und gelten als Formalisierung der drei Sätze Vater, Mutter und Tochter

$$(S \vee N) \wedge [(\neg S \vee N) \wedge (P \vee \neg S)] \wedge (\neg N \vee P).$$

Durch Ausklammern von $\neg S$ in der mittleren Unterformel wird die Formel in einem Schritt in CNF übernommen. Die Nummerierung der Klauseln mit tiefgestellten Indizes ergibt

$$KB \vee (S \vee N) \wedge 1 \vee (\neg S) \wedge 2 \vee (P \vee N) \wedge 3 \vee (\neg N \vee P) \wedge 4.$$

Jetzt beginnen wir mit dem Auflösungsbeweis, zunächst noch ohne Abfrage Q. Ein Ausdruck der Form „Res(m, n): Klauselk“ bedeutet, dass Klausel m durch Auflösung von Klausel n erhalten wird und die Nummer k hat.

$$\text{Res}(1, 2): (N) \wedge 5$$

$$\text{Res}(3, 4): (P) \wedge 6$$

$$\text{Res}(1, 4): (S \vee P) \wedge 7$$

Wir hätten Klausel P auch aus Res(4, 5) oder Res(2, 7) ableiten können. Jeder weitere Lösungsschritt würde zur Ableitung bereits vorhandener Klauseln führen. Da es die Ableitung des Leersatzes nicht zulässt, wurde somit gezeigt, dass die Wissensbasis nicht widersprüchlich ist. Bisher haben wir N und P abgeleitet. Um zu zeigen, dass $\neg S$ gilt, fügen wir die Klausel (S)8 als negierte Abfrage zur Menge der Klauseln hinzu. Mit dem Auflösungsschritt

$$\text{Res}(2, 8): () \wedge 9$$

Der Beweis ist vollständig. Somit gilt $\neg S \vee N \vee P$. Die „charmante englische Familie“ stammt offenbar aus Newcastle, hat in Paris Halt gemacht, reist aber nicht nach Spanien.

Beispiel 2.3 Logikrätsel Nummer 28 aus [Ber89] mit dem Titel „Der Hochsprung“ lautet

Drei Mädchen üben Hochsprung für ihre Abschlussprüfung im Sportunterricht. Die Messlatte ist auf 1,20 Meter festgelegt. „Ich wette“, sagt das erste Mädchen zum zweiten, „dass ich es schaffen werde, wenn und nur wenn du es nicht tust.“ Wenn das zweite Mädchen dasselbe zum dritten sagte, das wiederum dasselbe zum ersten sagte, wäre es dann möglich, dass alle drei ihre Wetten gewinnen?

Wir zeigen durch den Beweis durch Beschluss, dass nicht alle drei ihre Wetten gewinnen können.

Formalisierung:

Der Sprung des ersten Mädchens gelingt:	Wette des ersten Mädchens: $(A \leftrightarrow \neg B)$
A, der Sprung des zweiten Mädchens gelingt:	$\neg B$), Wette des zweiten Mädchens: $(B \leftrightarrow \neg C)$
B, der Sprung des dritten Mädchens gelingt: C.	$\neg C$), Wette des dritten Mädchens: $(C \leftrightarrow \neg A)$

Behauptung: Die drei können nicht alle ihre Wetten gewinnen:

$$Q \leftrightarrow \neg((A \leftrightarrow \neg B) \wedge (B \leftrightarrow \neg C) \wedge (C \leftrightarrow \neg A))$$

Es muss nun durch Resolution gezeigt werden, dass $\neg Q$ unerfüllbar ist.

Umwandlung in CNF: Erste Mädchenwette:

$$(A \leftrightarrow \neg B) \wedge (A \leftrightarrow \neg B) \wedge (\neg B \leftrightarrow A) \wedge (\neg A \leftrightarrow \neg B) \wedge (A \leftrightarrow B)$$

Die Wetten der anderen beiden Mädchen durchlaufen analoge Transformationen und wir erhalten die negierte Behauptung

$$\neg Q \leftrightarrow (\neg A \leftrightarrow \neg B) \wedge (A \leftrightarrow B) \wedge (\neg B \leftrightarrow \neg C) \wedge (B \leftrightarrow C) \wedge (\neg C \leftrightarrow \neg A) \wedge (C \leftrightarrow A)$$

Von dort leiten wir mittels Resolution den Leersatz ab:

$$\text{Res}(1, 6): (C \leftrightarrow \neg B)$$

$$\text{Res}(4, 7): (C)$$

$$\text{Res}(2, 5): (B \leftrightarrow \neg C)$$

$$\text{Res}(3, 9): (\neg C)$$

$$\text{Res}(8, 10): ()$$

Somit ist die Behauptung bewiesen.

2.5 Hornklauseln

Ein Satz in konjunktiver Normalform enthält positive und negative Literale und kann in der Form dargestellt werden

$$(\neg A_1 \vee \dots \vee \neg A_m \wedge B_1 \vee \dots \vee B_n)$$

mit den Variablen $A_1, \dots, A_m$ und $B_1, \dots, B_n$. Dieser Satz kann in zwei einfachen Schritten in die entsprechende Form umgewandelt werden

$$A_1 \rightarrow \dots \rightarrow A_m \rightarrow B_1 \rightarrow \dots \rightarrow B_n.$$

Diese Implikation enthält die Prämisse, eine Konjunktion von Variablen, und die Schlussfolgerung, eine Disjunktion von Variablen. Zum Beispiel: „*Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren oder arbeite.*“ ist ein Satz dieser Form. Der Empfänger dieser Nachricht weiß mit Sicherheit, dass der Absender nicht schwimmen geht.

Eine deutlich klarere Aussage wäre: „*Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren.*“ Der Empfänger weiß es jetzt definitiv. Daher nennen wir Klauseln mit höchstens einem positiven Literal definite Klauseln. Diese Klauseln haben den Vorteil, dass sie nur eine Schlussfolgerung zulassen und dadurch deutlich einfacher zu interpretieren sind. Viele Beziehungen können durch Klauseln dieser Art beschrieben werden. Wir definieren daher

Definition 2.10 Klauseln mit höchstens einem positiven Literal der Form

$$(\neg A_1 \rightarrow \dots \rightarrow \neg A_m \rightarrow B) \text{ oder } (\neg A_1 \rightarrow \dots \rightarrow \neg A_m) \text{ oder } B$$

oder gleichwertig)

$$A_1 \rightarrow \dots \rightarrow A_m \rightarrow B \text{ oder } A_1 \rightarrow \dots \rightarrow A_m \rightarrow f \text{ oder } B.$$

werden **Hornsätze** (nach ihrem Erfinder) genannt. Eine Klausel mit einem einzigen positiven Literal ist eine **Tatsache**. In Sätzen mit negativem und einem positiven Literal wird das positive Literal als **Kopf bezeichnet**.

Um die Darstellung von Horn-Klauseln besser zu verstehen, kann der Leser sie aus den Definitionen der Äquivalenzen ableiten, die wir derzeit verwendet haben (Übung 2.12 auf Seite 30).

Hornsätze sind nicht nur im täglichen Leben, sondern auch im formalen Denken einfacher zu handhaben, wie wir im folgenden Beispiel sehen können. Die Wissensbasis soll aus den folgenden Klauseln bestehen (das „ $\rightarrow$ “, das die Klauseln bindet, wird hier und im folgenden Text weggelassen):

(schönes_Wetter)¹

(Schneefall)²

(Schneefall $\rightarrow$ Schnee)³

(schönes Wetter $\rightarrow$ Schnee $\rightarrow$ Skifahren)⁴

Wenn wir nun wissen wollen, ob *das Skifahren* hält, lässt sich dies leicht ableiten. Als Inferenzregel genügt hier ein leicht verallgemeinerter Modus Ponens:

$$\frac{A1 \dot{\vee} \dots \dot{\vee} A_m, A1 \dot{\vee} \dots \dot{\vee} A_m \dot{\vee} B}{B} .$$

Der Beweis für „*Skifahren*“ hat die folgende Form (MP(i1,...,ik) stellt die Anwendung des Modus Ponens auf die Klauseln i1 bis ik dar :

$$\begin{aligned} \text{MP}(2, 3): & (\text{Schnee})5 \text{ MP}(1, \\ & 5, 4): (\text{Skifahren})6. \end{aligned}$$

Mit Modus Ponens erhalten wir einen vollständigen Kalkül für Formeln, die aus aussagenlogischen Hornsätzen bestehen. Bei großen Wissensbeständen kann der Modus Ponens jedoch viele unnötige Formeln ableiten, wenn man mit den falschen Sätzen beginnt. Daher ist es in vielen Fällen besser, eine Berechnung zu verwenden, die mit der Abfrage beginnt und rückwärts arbeitet, bis die Fakten vorliegen. Solche Systeme werden als *Rückwärtsverkettung bezeichnet*, im Gegensatz zu *Vorwärtsverkettungssystemen*, die mit Fakten beginnen und schließlich die Abfrage ableiten, wie im obigen Beispiel mit dem Modus Ponens.

Für die Rückwärtsverkettung von Horn-Klauseln wird *die SLD-Auflösung* verwendet. SLD steht für „*Selection Rule Driven Linear Resolution for Definite Clauses*“. Im obigen Beispiel erweitert um die negierte Abfrage (*Skifahren* $\dot{\vee}$ f)

$$\begin{aligned} & (\text{schönes\_Wetter})1 \\ & (\text{Schneefall})2 \\ & (\text{Schneefall} \dot{\vee} \text{Schnee})3 \\ & (\text{schönes\_Wetter} \dot{\vee} \text{Schnee} \dot{\vee} \text{Skifahren})4 \\ & (\text{Skifahren} \dot{\vee} \text{f})5 \end{aligned}$$

Wir führen die SLD-Lösung beginnend mit den Lösungsschritten durch, die sich aus dieser Klausel ergeben

$$\begin{aligned} \text{Res}(5, 4): & (\text{schönes\_Wetter} \dot{\vee} \text{Schnee} \dot{\vee} \text{f})6 \\ \text{Res}(6, 1): & (\text{Schnee} \dot{\vee} \text{f})7 \\ \text{Res}(7, 3): & (\text{Schneefall} \dot{\vee} \text{f})8 \\ \text{Res}(8, 2): & () \end{aligned}$$

und leiten Sie einen Widerspruch mit dem leeren Satz ab. Hier können wir leicht die „*lineare Auflösung*“ erkennen, was bedeutet, dass die weitere Verarbeitung immer für die aktuell abgeleitete Klausel erfolgt. Dies führt zu einer starken Reduzierung des Suchraums. Darüber hinaus werden die Literale der aktuellen Klausel immer in einer festen Reihenfolge (z. B. von rechts nach links) verarbeitet („*Selection Rule Driven*“). Die Literale der aktuellen Klausel werden aufgerufen

Unterziel. Die Literale der negierten Abfrage sind die **Ziele**. Die Inferenzregel für einen Schritt lautet

$$\frac{A_1 \dot{\vee} \dots \dot{\vee} A_m \dot{\vee} B_1, B_1 \dot{\vee} B_2 \dot{\vee} \dots \dot{\vee} B_n \dot{\vee} f}{A_1 \dot{\vee} \dots \dot{\vee} A_m \dot{\vee} B_2 \dot{\vee} \dots \dot{\vee} B_n \dot{\vee} f} .$$

Vor der Anwendung der Inferenzregel müssen $B_1, B_2, \dots, B_n$ – die aktuellen Teilziele – bewiesen werden. Nach der Anwendung wird B_1 durch das neue Teilziel $A_1 \dot{\vee} \dots \dot{\vee} A_m$ ersetzt. Um zu zeigen, dass B_1 wahr ist, müssen wir nun zeigen, dass $A_1 \dot{\vee} \dots \dot{\vee} A_m$ wahr sind. Dieser Vorgang wird so lange fortgesetzt, bis die Liste der Unterziele der aktuellen Klauseln (der sogenannte *Zielstapel*) leer ist. Damit wurde ein Widerspruch festgestellt. Wenn es für ein Unterziel $\neg B_i$ keinen Satz mit dem Komplementärliteral B_i als Satzkopf gibt, endet der Beweis und es kann kein Widerspruch gefunden werden. Die Anfrage ist somit unbeweisbar.

Die SLD-Auflösung spielt in der Praxis eine wichtige Rolle, da Programme in der logischen Programmiersprache PROLOG aus prädikatenlogischen Horn-Klauseln bestehen und ihre Verarbeitung mittels SLD-Auflösung erfolgt (siehe Übung 2.13 auf Seite 30, bzw. Kap. 5).

2.6 Berechenbarkeit und Komplexität

Die Wahrheitstabellenmethode stellt als einfachstes semantisches Beweissystem für die Aussagenlogik einen Algorithmus dar, der jedes Modell einer beliebigen Formel in endlicher Zeit bestimmen kann. Somit sind die Mengen unerfüllbarer, erfüllbarer und gültiger Formeln entscheidbar. Die Rechenzeit der Wahrheitstabellenmethode zur Erfüllbarkeit wächst im ungünstigsten Fall exponentiell mit der Anzahl n der Variablen, Nr. 2 da die Wahrheitstabelle Zeilen hat. Eine Optimierung, die Methode der *semantischen Bäume*, vermeidet die Betrachtung von Variablen, die nicht in Klauseln vorkommen, und spart so in vielen Fällen Rechenzeit, im schlimmsten Fall ist sie aber ebenfalls exponentiell.

Bei der Auflösung wächst im ungünstigsten Fall die Anzahl der abgeleiteten Sätze exponentiell mit der Anzahl der Anfangssätze. Um zwischen den beiden Verfahren zu entscheiden, können wir daher die Faustregel verwenden, dass bei vielen Klauseln mit wenigen Variablen die Wahrheitstabellenmethode vorzuziehen ist und bei wenigen Klauseln mit vielen Variablen die Auflösung wahrscheinlich schneller abgeschlossen wird.

Es bleibt die Frage: Kann der Beweis in der Aussagenlogik schneller gehen? Gibt es bessere Algorithmen? Die Antwort: wahrscheinlich nicht. Schließlich hat S. Cook, der Begründer der Komplexitätstheorie, gezeigt, dass das 3-SAT-Problem NP-vollständig ist. 3-SAT ist die Menge aller CNF-Formeln, deren Klauseln genau drei Literale haben. Somit ist klar, dass es für 3-SAT wahrscheinlich (modulo des P/NP-Problems) keinen polynomialen Algorithmus und daher wahrscheinlich auch keinen allgemeinen gibt. Für Horn-Klauseln gibt es jedoch einen Algorithmus, bei dem die Rechenzeit zum Testen der Erfüllbarkeit nur linear mit zunehmender Anzahl an Literalen in der Formel wächst.

2.7 Anwendungen und Einschränkungen

Theorembeweiser für Aussagenlogik gehören zum alltäglichen Werkzeugset des Entwicklers in der digitalen Technologie. Zu diesen Aufgaben gehören beispielsweise die Verifizierung digitaler Schaltkreise und die Generierung von Testmustern zum Testen von Mikroprozessoren in der Fertigung. Als Datenstruktur zur Verarbeitung aussagenlogischer Formeln werden auch spezielle Beweissysteme eingesetzt, die mit binären Entscheidungen arbeiten.

In der KI wird Aussagenlogik in einfachen Anwendungen eingesetzt. Einfache Expertensysteme können beispielsweise durchaus mit Aussagenlogik arbeiten. Allerdings müssen die Variablen alle diskret sein und nur wenige Werte aufweisen, und es dürfen keine Kreuzbeziehungen zwischen den Variablen bestehen. Komplexe logische Zusammenhänge lassen sich mit der Prädikatenlogik wesentlich eleganter ausdrücken.

Die probabilistische Logik ist eine sehr interessante und aktuelle Kombination aus Aussagenlogik und probabilistischer Berechnung, die die Modellierung unsicheren Wissens ermöglicht. Es wird in Kap. ausführlich behandelt. 7. In diesem Kapitel wird auch die Fuzzy-Logik besprochen, die unendlich viele Wahrheitswerte zulässt.

2.8 Übungen

Übung 2.1 Geben Sie eine Backus-Naur-Formengrammatik für die Syntax von Aussagen an Logik.

Aufgabe 2.2 Zeigen Sie, dass die folgenden Formeln Tautologien

sind: (a) $\neg(A \vee B) \vee \neg A \vee \neg B$ (b)

$A \vee B \vee \neg B \vee \neg A$ (c) $((A \vee B)$

$\vee (B \vee A)) \vee (A \vee B)$ (d) $(A \vee B) \vee (\neg B \vee C) \vee (A$

$\vee C)$

Aufgabe 2.3 Wandeln Sie die folgenden Formeln in konjunktive Normalform um: (a)

$A \vee B$ (b) A

$\vee B \vee A \vee B$ (c) $A \vee (A$

$\vee B) \vee B$

Aufgabe 2.4 Überprüfen Sie die folgenden Aussagen auf Erfüllbarkeit bzw.

Gültigkeit. (a) $(\text{play\_lottery} \vee \text{six\_right}) \vee$

Gewinner (b) $(\text{play\_lottery} \vee \text{six\_right} \vee (\text{six\_right} \vee \text{win})) \vee \text{win}$

(c) $\neg(\text{gas\_in\_tank} \vee (\text{gas\_in\_tank} \vee \neg \text{car\_starts})) \vee \neg \text{car\_starts}$

Übung 2.5 Programmieren Sie mit der Programmiersprache Ihrer Wahl einen

Theorembeweis für Aussagenlogik unter Verwendung der Wahrheitstabellenmethode für Formeln in konjunktiver Normalform. Um eine kostspielige Syntaxprüfung der Formeln zu vermeiden, können Sie Klauseln als Listen oder Mengen von Literalen und die Formeln als Listen oder Mengen von Klauseln darstellen. Das Programm soll anzeigen, ob die Formel unerfüllbar, erfüllbar oder wahr ist, und die Anzahl unterschiedlicher Interpretationen und Modelle.

Aufgabe 2.6

(a) Zeigen Sie, dass Modus Ponens eine gültige Inferenzregel ist, indem Sie zeigen, dass $A \rightarrow (A \rightarrow B) \vdash B$. (b)

Zeigen Sie, dass die Resolutionsregel (2.1) eine gültige Inferenzregel ist.

Üb Aufgabe 2.7 Zeigen Sie durch Anwendung der Auflösungsregel, dass im Konjunktivnormal Form ist die leere Klausel gleichbedeutend mit der falschen Aussage.

Ü Übung 2.8 Zeigen Sie, dass man mit Resolution jede beliebige Klausel daraus „ableiten“ kann eine Wissensbasis, die einen Widerspruch enthält.

Aufgabe 2.9 Formalisieren Sie die folgenden logischen Funktionen mit den logischen Operatoren und zeigen Sie, dass Ihre Formel gültig ist. Präsentieren Sie das Ergebnis in CNF. (a) Die XOR-Operation (exklusiv oder)

zwischen zwei Variablen. (b) Die Aussage, dass mindestens zwei der drei Variablen A,B,C wahr sind.

Ü Übung 2.10 Lösen Sie den folgenden Fall mit Hilfe eines Auflösungsbeweises: „Wenn der Täter einen Komplizen hatte, dann kam er mit einem Auto.“ Der Täter hatte keinen Komplizen und keinen Schlüssel, oder er hatte den Schlüssel und einen Komplizen. Der Verbrecher hatte den Schlüssel. Ist der Verbrecher mit einem Auto gekommen oder nicht?“

Aufgabe 2.11 Zeigen Sie durch Auflösung, dass die Formel aus

(a) Aufgabe 2.2(d) eine Tautologie ist.

(b) Aufgabe 2.4(c) ist nicht erfüllbar.

Aufgabe 2.12 Beweisen Sie die folgenden Äquivalenzen, die für die Arbeit mit Hornsätzen wichtig sind: (a)

$(\neg A_1 \rightarrow \dots \rightarrow \neg A_m \rightarrow B) \rightarrow A_1 \rightarrow \dots \rightarrow A_m \rightarrow B$ (b) $(\neg A_1 \rightarrow \dots \rightarrow \neg A_m) \rightarrow$

$A_1 \rightarrow \dots \rightarrow A_m \rightarrow f$ (c) $A \rightarrow w \rightarrow A$

Aufgabe 2.13 Zeigen Sie durch SLD-Auflösung, dass die folgende Horn-Klauselmengue unerfüllbar ist.

(A)1	(D)4	$(A \rightarrow D \rightarrow G)7$
(B)2	(E)5	$(C \rightarrow F \rightarrow E \rightarrow H)8$
(C)3	$(A \rightarrow B \rightarrow C \rightarrow F)6$	$(H \rightarrow f)9$

Ü Übung 2.14 In Abschn. 2.6 heißt es: „Damit ist klar, dass es wahrscheinlich (modulo des P/NP-Problems) keinen polynomialen Algorithmus für 3-SAT gibt und daher wahrscheinlich auch keinen allgemeinen.“ Begründen Sie das „wahrscheinlich“ in diesem Satz.

Prädikatenlogik erster Ordnung

3

Viele praktisch relevante Probleme lassen sich nicht oder nur sehr umständlich in der Sprache der Aussagenlogik formulieren, wie wir am folgenden Beispiel leicht erkennen können. Die Aussage

„Roboter 7 befindet sich an der xy-Position (35, 79)“

kann tatsächlich direkt als aussagenlogische Variable verwendet werden

„Robot_7_is_situated_at_xy_position_(35, 79)“

für das Denken mit Aussagenlogik, aber das Denken mit dieser Art von Aussage ist sehr unbequem. Angenommen, 100 dieser Roboter können irgendwo auf einem Raster von 100×100 Punkten anhalten. Um jede Position jedes Roboters zu beschreiben, bräuchten wir $100 \cdot 100 \cdot 100 = 1.000.000 = 10^6$ verschiedene Variablen. Die Definition von Beziehungen zwischen Objekten (hier Robotern) wird wirklich schwierig. Die Beziehung

„Roboter A steht rechts von Roboter B.“

ist semantisch nichts anderes als eine Menge von Paaren. Von den 10.000 möglichen x-Koordinatenpaaren gibt es $(99 \cdot 98)/2 = 4851$ geordnete Paare. Zusammen mit allen 10.000 Kombinationen möglicher y-Werte für beide Roboter ergibt sich $(100 \cdot 99) = 9900$ für Mulas des Typs

Robot_7_is_to_the_right_of_robot_12 $\dot{\vee}$

Robot_7_is_situated_at_xy_position_(35, 79) $\dot{\vee}$

Robot_12_is_situated_at_xy_position_(10, 93) $\dot{\vee}$...

Definieren Sie diese Beziehungen, jeweils mit $(104)^2 \cdot 0,485 = 0,485 \cdot 108$ Alternativen auf der rechten Seite. In der Prädikatenlogik erster Ordnung können wir hierfür ein Prädikat *Position(Zahl, xPosition, yPosition)* definieren. Die obige Relation muss nicht mehr als eine große Anzahl von Paaren aufgezählt werden, sondern wird abstrakt mit einer Formregel beschrieben

$\dot{\forall} u \dot{\forall} v \text{ is\_further\_right}(u, v) \dot{\vee} \dot{\exists} x u$

$\dot{\forall} y u \dot{\forall} x v \dot{\forall} y v \text{ position}(u, x u, y u) \dot{\vee} \text{ position}(v, x v, y v) \dot{\vee} x u > x v,$

Wobei $\dot{\forall} u$ als „für jedes u“ und $\dot{\forall} v$ als „es existiert v“ gelesen wird.

In diesem Kapitel definieren wir die Syntax und Semantik der *Prädikatenlogik erster Ordnung* (PL1), zeigen, dass viele Anwendungen mit dieser Sprache modelliert werden können und zeigen, dass es für diese Sprache einen vollständigen und fundierten Kalkül gibt.

3.1 Syntax

Zunächst festigen wir die syntaktische Struktur von Begriffen.

Definition 3.1 Es sei V eine Menge von Variablen, K eine Menge von Konstanten und F eine Menge von Funktionssymbolen. Die Mengen V und F sind paarweise disjunkt. Wir definieren die Menge der *Terme* rekursiv:

- Alle Variablen und Konstanten sind (atomare) Terme.
- Wenn $t_1, \dots, t_n$ Terme sind und f ein n -stelliges Funktionssymbol, dann ist $f(t_1, \dots, t_n)$ auch ein Begriff.

Einige Beispiele für Begriffe sind $f(\sin(\ln(3)), \exp(x))$ und $g(g(g(x)))$. In der Lage sein stellen logische Beziehungen zwischen Begriffen her, wir bilden Formeln aus Begriffen.

Definition 3.2 Sei P eine Menge von Prädikatsymbolen. *Formeln der Prädikatenlogik* werden wie folgt aufgebaut:

- Wenn $t_1, \dots, t_n$ Terme sind und p ein n -stelliges Prädikatssymbol, dann ist $p(t_1, \dots, t_n)$ eine (atomare) Formel.
- Wenn A und B Formeln sind, dann sind $\neg A$, (A) , $A \vee B$, $A \wedge B$, $A \rightarrow B$, $A \leftrightarrow B$ ebenfalls Formeln.
- Wenn x eine Variable und A eine Formel ist, dann sind $\forall x A$ und $\exists x A$ ebenfalls Formeln. $\forall$ ist der universelle Quantor und $\exists$ der Existenzquantor.
- $p(t_1, \dots, t_n)$ und $\neg p(t_1, \dots, t_n)$ heißen *Literale*.
- Formeln, in denen jede Variable im Gültigkeitsbereich eines Quantors liegt, werden *Sätze erster Ordnung* oder *geschlossene Formeln* genannt. Variablen, die nicht im Gültigkeitsbereich eines Quantors liegen, werden *freie Variablen* genannt.
- Die Definitionen 2.8 (CNF) und 2.10 (Horn-Klauseln) gelten analog für Formeln prädikatslogischer Literale.

In Tabelle 3.1 auf Seite 33 sind mehrere Beispiele für PL1-Formeln aufgeführt mit ihren intuitiven Interpretationen.

Tabelle 3.1 Beispiele für Formeln in der Prädikatenlogik erster Ordnung. Bitte beachten Sie, dass es sich hier um eine *Mutter* handelt
Funktionssymbol

Formel	Beschreibung
$\forall x \text{Frosch}(x) \rightarrow \text{grün}(x) \forall x$	Alle Frösche sind grün
$\text{Frosch}(x) \rightarrow \text{braun}(x) \rightarrow \text{groß}(x) \forall x$	Alle braunen Frösche sind groß
$\text{mag}(x, \text{Kuchen})$	Jeder mag Kuchen
$\neg \forall x \text{Likes}(x, \text{Kuchen})$	Nicht jeder mag Kuchen
$\neg \exists x \text{Likes}(x, \text{Kuchen})$	Niemand mag Kuchen
$\exists x \exists y \text{mag}(y, x) \forall x$	Es gibt etwas, das jedem gefällt
$\exists y \text{mag}(x, y) \forall x \forall y$	Es gibt jemanden, der alles mag
$\text{mag}(y, x) \forall x \forall y$	Alles wird von jemandem geliebt
$\text{mag}(x, y) \forall x$	Jeder mag etwas
$\text{Kunde}(x) \rightarrow \text{mag}(\text{bob}, x) \forall x \text{Kunde}(x)$	Bob mag jeden Kunden
$\exists \text{Likes}(x, \text{Bob}) \forall x \text{Bäcker}(x) \rightarrow \exists y$	Es gibt einen Kunden, den Bob mag
$\text{Kunde}(y) \rightarrow \text{mag}(x, y)$ Es gibt einen Bäcker, der alle seine mag Kunden	
$\forall x \text{älter}(\text{Mutter}(x), x) \forall x$	Jede Mutter ist älter als ihr Kind
$\text{älter}(\text{Mutter}(\text{Mutter}(x)), x)$	Jede Großmutter ist älter als sie Kind der Tochter
$\forall x \forall y \exists z \text{rel}(x, y) \rightarrow \text{rel}(y, z) \rightarrow \text{rel}(x, z)$	<i>rel</i> ist eine transitive Beziehung

3.2 Semantik

In der Aussagenlogik wird jeder Variablen durch eine Interpretation direkt ein Wahrheitswert zugeordnet. In der Prädikatenlogik wird die Bedeutung von Formeln rekursiv über die definiert Aufbau der Formel, indem wir zunächst Konstanten, Variablen und Funktion zuweisen Symbole zu Objekten in der realen Welt.

Definition 3.3 Eine *Interpretation* I ist definiert als

- Eine Abbildung von der Menge der Konstanten und Variablen $K \cup V$ auf eine Menge W von Namen von Objekten in der Welt.
- Eine Zuordnung von der Menge der Funktionssymbole zur Menge der Funktionen im Welt. Jedem n -stelligen Funktionssymbol ist eine n -stellige Funktion zugeordnet.
- Eine Abbildung von der Menge der Prädikatsymbole auf die Menge der Beziehungen im Welt. Jedem n -stelligen Prädikatsymbol ist eine n -stellige Relation zugeordnet.

Beispiel 3.1 Es seien c_1, c_2, c_3 Konstanten, „*plus*“ ein zweistelliges Funktionssymbol und „*gr*“ ein zweistelliges Prädikatsymbol. Die Wahrheit der Formel

$$F \rightarrow \text{gr}(\text{plus}(c_1, c_3), c_2)$$

hängt von der Interpretation I ab. Wir wählen zunächst die folgende naheliegende Interpretation der Konstanten, der Funktion und der Prädikate in den natürlichen Zahlen:

$$I_1: c_1 \checkmark 1, c_2 \checkmark 2, c_3 \checkmark 3, \text{plus} \checkmark +, \text{gr} \checkmark >$$

Somit wird die Formel abgebildet

$$1 + 3 > 2, \text{ oder nach Auswertung } 4 > 2.$$

Die Größer-als-Beziehung auf der Menge $\{1, 2, 3, 4\}$ ist die Menge aller Zahlenpaare (x, y) mit $x > y$, d. h. die Menge $G = \{(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)\}$. Da $(4, 2) \checkmark G$, ist die Formel F unter der Interpretation I_1 wahr. Wenn wir jedoch die Interpretation wählen

$$I_2: c_1 \checkmark 2, c_2 \checkmark 3, c_3 \checkmark 1, \text{plus} \checkmark \checkmark, \text{gr} \checkmark >,$$

wir erhalten

$$2 \checkmark 1 > 3 \text{ oder } 1 > 3.$$

Das Paar $(1, 3)$ ist kein Mitglied von G . Die Formel F ist unter der Interpretation I_2 falsch. Offensichtlich hängt die Wahrheit einer Formel in PL1 von der Interpretation ab. Nach dieser Vorschau definieren wir nun die Wahrheit.

Definition 3.4 •

Eine atomare Formel $p(t_1, \dots, t_n)$ ist *wahr* (oder *gültig*) unter der Interpretation I , wenn nach Interpretation und Bewertung aller Terme $t_1, \dots, t_n$ und Interpretation des Prädikats p bis die n -Stellen-Relation r , das gilt

$$(I(t_1), \dots, I(t_n)) \checkmark r.$$

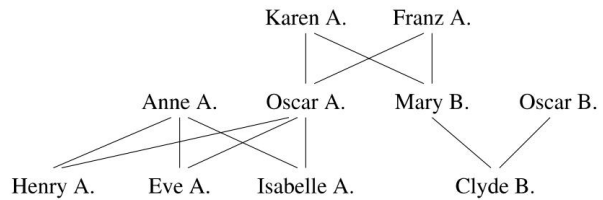
- Die Wahrheit quantifizierender Formeln folgt aus der Wahrheit atomarer Formeln – wie in der Aussagenrechnung – durch die Semantik der in Tabelle 2.1 auf Seite 17 definierten logischen Operatoren. • Eine Formel

$\checkmark x F$ ist unter der Interpretation I genau dann wahr, wenn sie gilt ist wahr bei einer willkürlichen Änderung der Interpretation für die Variable x (und nur für x)

- Eine Formel $\checkmark x F$ ist unter der Interpretation I genau dann wahr, wenn es für x eine Interpretation gibt, die die Formel wahr macht.

Die Definitionen der semantischen Äquivalenz von Formeln für die Konzepte erfüllbar, wahr, unerfüllbar und Modell sowie semantische Folgerung (Definitionen 2.4, 2.5, 2.6) werden unverändert aus der Aussagenlogik in die Prädikatenlogik übernommen.

Abb. 3.1 Ein Stammbaum.
Die Kanten, die von Clyde B. aufwärts zu Mary B. und Oscar B. reichen, stellen das Element (Clyde B., Mary B., Oscar B.) als Kindbeziehung dar



Satz 3.1 Die Sätze 2.2 (Abzugssatz) und 2.3 (Widerspruchsbeweis) gelten analog für PL1.

Beispiel 3.2 Der in Abb. 3.1 dargestellte Stammbaum stellt die Beziehung grafisch (auf der semantischen Ebene) dar

$\text{Kind} = \{(\text{Oscar A.}, \text{Karen A.}, \text{Frank A.}), (\text{Mary B.}, \text{Karen A.}, \text{Frank A.}), (\text{Henry A.}, \text{Anne A.}, \text{Oscar A.}), (\text{Eve A.}, \text{Anne A.}, \text{Oscar A.}), (\text{Isabelle A.}, \text{Anne A.}, \text{Oscar A.}), (\text{Clyde B.}, \text{Mary B.}, \text{Oscar B.})\}$

Das Tripel (Oscar A., Karen A., Frank A.) steht beispielsweise für die Aussage „Oscar A. ist ein Kind von Karen A. und Frank A.“. Aus den Namen lesen wir die Ein-Stellen-Relation ab

$\text{Weiblich} = \{\text{Karen A.}, \text{Anne A.}, \text{Mary B.}, \text{Eve A.}, \text{Isabelle A.}\}$

der Frauen. Wir wollen nun Formeln für familiäre Beziehungen aufstellen. Zuerst definieren wir ein dreistelliges Prädikat $\text{child}(x, y, z)$ mit der Semantik

$$I(\text{Kind}(x, y, z)) = w \text{ } \checkmark \text{ } (I(x), I(y), I(z)) \text{ } \checkmark \text{ } \text{Kind}.$$

Unter der Interpretation $I(\text{oscar}) = \text{Oscar A.}$, $I(\text{eve}) = \text{Eve A.}$, $I(\text{anne}) = \text{Anne A.}$ gilt auch, dass $\text{child}(\text{eve}, \text{anne}, \text{oscar})$. Damit $\text{child}(\text{eve}, \text{oscar}, \text{anne})$ wahr ist, benötigen wir, with

$$\checkmark x \checkmark y \checkmark z \text{ Kind}(x, y, z) \text{ } \checkmark \text{ Kind}(x, z, y),$$

Symmetrie des Prädikatkindes in den letzten beiden Argumenten. Für weitere Definitionen verweisen wir auf Übung 3.1 auf Seite 54 und definieren das Prädikat *Deszendent* rekursiv als

$$\checkmark x \checkmark y \text{ Nachkomme}(x, y) \text{ } \checkmark \checkmark z \text{ Kind}(x, y, z) \text{ } \checkmark \\ (\checkmark u \checkmark v \text{ Kind}(x, u, v) \text{ } \checkmark \text{ Nachkomme}(u, y)).$$

Jetzt bauen wir eine kleine Wissensbasis mit Regeln und Fakten auf. Lassen

$KB \vdash \text{weiblich}(\text{Karen}) \vdash \text{weiblich}(\text{Anne}) \vdash \text{weiblich}(\text{Mary})$
 $\vdash \text{weiblich}(\text{Eve}) \vdash \text{weiblich}(\text{Isabelle})$
 $\vdash \text{Kind}(\text{Oscar}, \text{Karen}, \text{Franz}) \vdash \text{Kind}(\text{Mary}, \text{Karen}, \text{Franz})$
 $\vdash \text{Kind}(\text{Eve}, \text{Anne}, \text{Oscar}) \vdash \text{Kind}(\text{Henry}, \text{Anne}, \text{Oscar})$
 $\vdash \text{Kind}(\text{Isabelle}, \text{Anne}, \text{Oscar}) \vdash \text{Kind}(\text{Clyde}, \text{Mary}, \text{Oscar})$
 $\vdash (\forall x \forall y \forall z \text{ Kind}(x, y, z) \vdash \text{Kind}(x, z, y))$
 $\vdash (\forall x \forall y \text{ Nachkomme}(x, y) \vdash \forall z \text{ Kind}(x, y, z))$
 $\vdash (\forall u \forall v \text{ Kind}(x, u, v) \vdash \text{Nachkomme}(u, y))$.

Wir können nun beispielsweise fragen, ob die Propositionen $\text{child}(\text{eve}, \text{oscar}, \text{anne})$ oder $\text{Descendent}(\text{eve}, \text{franz})$ ableitbar sind. Dazu benötigen wir einen Kalkül.

3.2.1 Gleichheit

Um Begriffe vergleichen zu können, ist Gleichheit eine sehr wichtige Relation in der Prädikatenlogik. Die Gleichheit von Termen in der Mathematik ist eine Äquivalenzrelation, also reflexiv, symmetrisch und transitiv. Wenn wir Gleichheit in Formeln verwenden wollen, müssen wir entweder diese drei Attribute als Axiome in unsere Wissensbasis aufnehmen oder wir müssen Gleichheit in die Infinitesimalrechnung integrieren. Wir gehen den einfachen Weg und definieren ein Prädikat „=“, das abweichend von Definition 3.2 auf Seite 32 in der in der Mathematik üblichen Infix-Notation geschrieben wird. (Eine Gleichung $x = y$ könnte natürlich auch in der Form $\text{eq}(x, y)$ geschrieben werden.) Somit haben die Gleichheitsaxiome die Form

$$\begin{array}{ll}
 \forall x x = x & \text{(Reflexivität)} \\
 \forall y x = y \vdash y = x & \text{(Symmetrie)} \\
 \forall x \forall y \forall z x = y \vdash y = z \vdash x = z & \text{(Transitivität)}.
 \end{array} \tag{3.1}$$

Um die Einzigartigkeit der Funktionen zu gewährleisten, benötigen wir zusätzlich

$$\forall x \forall y x = y \vdash f(x) = f(y) \text{ (Substitutionsaxiom)} \tag{3.2}$$

für jedes Funktionssymbol. Analog benötigen wir für alle Prädikatsymbole

$$\forall x \forall y x = y \vdash p(x) \vdash p(y) \text{ (Substitutionsaxiom)}. \tag{3.3}$$

Auf ähnliche Weise formulieren wir auch andere mathematische Beziehungen, etwa die „<“-Relation (Übung 3.4 auf Seite 55).

Oft muss eine Variable durch einen Term ersetzt werden. Um dies richtig durchzuführen und Um es einfach zu beschreiben, geben wir die folgende Definition.

Definition 3.5 Wir schreiben $\tilde{y}[x/t]$ für die Formel, die sich ergibt, wenn wir jedes freie Vorkommen der Variablen x in $\tilde{y}$ durch den Term t ersetzen. Dabei lassen wir im Term t keine Variablen zu, die in $\tilde{y}$ quantifiziert sind. In diesen Fällen müssen Variablen umbenannt werden, um dies sicherzustellen.

Beispiel 3.3 Wenn in der Formel $\tilde{y}_{xx} = y$ die freie Variable y durch den Term $x + 1$ ersetzt wird, ist das Ergebnis $\tilde{y}_{xx} = x + 1$. Bei korrekter Substitution erhalten wir die Formel $\tilde{y}_{xx} = y + 1$, die hat eine ganz andere Semantik.

3.3 Quantoren und Normalformen

Nach Definition 3.4 auf Seite 34 ist die Formel $\tilde{y}_{xp}(x)$ genau dann wahr, wenn sie für alle Interpretationen der Variablen x wahr ist. Anstelle des Quantors könnte man $p(a_1) \tilde{y} \dots \tilde{y} p(a_n)$ für alle Konstanten $a_1 \dots a_n$ in K schreiben. Für $\tilde{y}_{xp}(x)$ könnte man $p(a_1) \tilde{y} \dots \tilde{y}$ schreiben (Panne). Daraus folgt mit dem Gesetz von de Morgan das

$$\tilde{y}_x \tilde{y} \tilde{y} \neg \tilde{y}_x \neg \tilde{y}.$$

Durch diese Äquivalenz sind universelle und existentielle Quantoren wechselseitig ersetzbar.

Beispiel 3.4 Die Aussage „Jeder möchte geliebt werden“ ist äquivalent zur Aussage „Niemand möchte nicht geliebt werden“.

Quantoren sind ein wichtiger Bestandteil der Ausdruckskraft der Prädikatenlogik. Für die automatische Inferenz in der KI sind sie jedoch störend, da sie die Struktur von Formeln komplexer machen und die Anzahl anwendbarer Inferenzregeln in jedem Schritt eines Beweises erhöhen. Unser nächstes Ziel ist es daher, für jede prädikatenlogische Formel eine äquivalente Formel in einer standardisierten Normalform mit möglichst wenigen Quantoren zu finden. Als ersten Schritt bringen wir universelle Quantoren an den Anfang der Formel und definieren sie so

Definition 3.6 Eine prädikatenlogische Formel $\tilde{y}$ liegt in *Pränex-Normalform* vor, wenn gilt:

- $\tilde{y} = Q_1 x_1 \dots Q_n x_n \tilde{y}$. • $\tilde{y}$ ist eine quantorenlose Formel. • $Q_i \in \{\forall, \exists\}$ für $i = 1, \dots, n$.

Vorsicht ist geboten, wenn eine quantifizierte Variable außerhalb ihres Quan-Bereichs erscheint
tifier, wie zum Beispiel x in

$$\forall x p(x) \wedge \exists x q(x).$$

Hier muss eine der beiden Variablen umbenannt werden, und zwar in

$$\forall x p(x) \wedge \exists y q(y)$$

Der Quantor kann leicht in den Vordergrund gebracht werden und wir erhalten als Ausgabe
die äquivalente Formel

$$\forall x \forall y p(x) \wedge q(y).$$

Wenn wir jedoch den Quantor korrekt in den Vordergrund stellen möchten

$$(\forall x p(x)) \wedge \exists y q(y) \tag{3.4}$$

Wir schreiben die Formel zunächst in der entsprechenden Form

$$\neg(\exists x p(x)) \wedge \exists y q(y).$$

Der erste universelle Quantor wird nun zu

$$(\forall x \neg p(x)) \wedge \exists y q(y)$$

und nun können die beiden Quantoren endlich nach vorne gezogen werden

$$\forall x \forall y \neg p(x) \wedge q(y),$$

was gleichbedeutend mit ist

$$\forall x \forall y p(x) \wedge q(y).$$

Wir sehen dann, dass wir in (3.4) nicht einfach beide Quantoren in den Vordergrund ziehen
können. Vielmehr müssen wir zunächst die Implikationen eliminieren, damit es keine
Verneinungen zu den Quantifizierern gibt. Generell gilt, dass wir Quantoren nur dann
herausziehen dürfen, wenn Negationen nur direkt auf atomaren Unterformeln existieren.

Beispiel 3.5 Wie in der Analysis allgemein bekannt ist, ist die Konvergenz einer Reihe $(a_n)_{n \in \mathbb{N}}$ zu
einem Grenzwert a definiert durch

$$\forall \epsilon > 0 \exists n_0 \in \mathbb{N} \forall n > n_0 |a_n - a| < \epsilon.$$

Mit der Funktion $\text{abs}(x)$ für $|x|$, $a(n)$ für a_n , $\text{minus}(x, y)$ für $x - y$ und den Prädikaten $\text{el}(x, y)$
für $x \in y$, $\text{gr}(x, y)$ für $x > y$ lautet die Formel

NORMALFORMTRANSFORMATION(Formel): 1.**Transformation in die Pränex-Normalform:****Transformation in die konjunktive Normalform (Satz 2.1):**

Eliminierung von Äquivalenzen.

Beseitigung von Implikationen.

Wiederholte Anwendung des de Morganschen Gesetzes und des Verteilungsgesetzes.

Umbenennen von Variablen bei Bedarf.

Universelle Quantoren herausrechnen.

2. Skolemisierung:

Ersatz existenziell quantifizierter Variablen durch neue Skolemfunktionen.

Löschung resultierender Universalquantoren.

Abb. 3.2 Transformation prädikatenlogischer Formeln in Normalform

Nun können wir den Existenzquantor (und damit auch den Universalquantor) in (3.5) auf Seite 39 eliminieren, indem wir die Skolem-Funktion $n0(\bar{y})$ einführen. Die skolemisierte Pränex- und Konjunktivnormalform von (3.5) auf Seite 39 lautet also

$$\neg gr(\bar{y}, 0) \vee \neg el(n0(\bar{y}), N) \vee \neg gr(n, n0(\bar{y})) \vee gr(\bar{y}, abs(minus(a(n), a))).$$

Durch Weglassen der Variablen $n0$ kann die Skolem-Funktion den Namen $n0$ erhalten.

Bei der Skolemisierung einer Formel in der Pränex-Normalform werden alle Existenzquantoren von außen nach innen eliminiert, wobei eine Formel der Form $\bar{y}x_1 \dots \bar{y}x_n \bar{y}y$ durch $\bar{y}x_1 \dots \bar{y}x_n \bar{y}[y/f(x_1, \dots, x_n)]$, wobei f möglicherweise nicht in $\bar{y}$ erscheint.

Liegt ein Existenzquantor weit außerhalb, etwa in $\bar{y}yp(y)$, dann muss y durch eine Konstante (also durch ein Nullstellen-Funktionssymbol) ersetzt werden.

Das Verfahren zur Transformation einer Formel in konjunktive Normalform ist in dem in Abb. 3.2 dargestellten Pseudocode zusammengefasst. Die Skolemisierung hat eine polynomielle Laufzeit in der Anzahl der Literale. Bei der Transformation in die Normalform kann die Anzahl der Literale in der Normalform exponentiell ansteigen, was zu einer exponentiellen Rechenzeit und einem exponentiellen Speicherverbrauch führen kann. Der Grund dafür ist die wiederholte Anwendung des Distributivgesetzes. Das eigentliche Problem, das sich aus einer Vielzahl von Klauseln ergibt, ist die kombinatorische Explosion des Suchraums für einen nachfolgenden Lösungsbeweis. Allerdings gibt es einen optimierten Transformationsalgorithmus, der nur polynomiell viele Literale erzeugt [Ede91].

3.4 Beweiskalküle

Für das Denken in der Prädikatenlogik wurden verschiedene Kalküle des natürlichen Denkens wie die Gentzen-Kalküle oder die Folgenrechnung entwickelt. Wie der Name schon sagt, sind diese Kalküle für die Anwendung durch Menschen gedacht, da die Inferenzregeln umfas-

Tabelle 3.2 Einfacher Beweis mit Modus Ponens und Quantoreneliminierung

WB:	1	Kind (Eve, Anne, Oscar)
WB:	2	$\forall x \forall y \forall z \text{ Kind}(x, y, z) \rightarrow \text{Kind}(x, z, y)$
$\forall E(2) : x/\text{eve}, y/\text{anne}, z/\text{oscar}$	3	$\text{kind}(\text{eve}, \text{anne}, \text{oscar}) \rightarrow \text{kind}(\text{eve}, \text{oscar}, \text{anne})$
MP(1, 3)	4	Kind (Eve, Oscar, Anne)

oder weniger intuitiv und die Kalküle funktionieren auf beliebigen PL1-Formeln. Im nächsten Abschnitt Wir werden uns hauptsächlich auf die Auflösungsrechnung konzentrieren, die in der Praxis die ist wichtigste effiziente, automatisierbare Berechnung für Formeln im Konjunktivnormal bilden. Hier werden wir anhand von Beispiel 3.2 auf Seite 35 einen sehr kleinen „natürlichen“ Beweis liefern. Wir verwenden die Inferenzregel

$$\frac{A, A \rightarrow B}{B} \quad (\text{modus ponens, MP}) \text{ und } \frac{\forall x A}{A[x/t]} \quad (\forall\text{-Eliminierung, } \forall E).$$

Der Modus Ponens ist bereits aus der Aussagenlogik bekannt. Beim Eliminieren Bei universellen Quantoren muss man bedenken, dass die quantifizierte Variable x sein muss ersetzt durch einen Grundterm t, also einen Term, der keine Variablen enthält. Der Beweis von child(eve, oscar, anne) aus einer entsprechend reduzierten Wissensbasis wird vorgestellt in Tabelle 3.2.

Die beiden Formeln der reduzierten Wissensbasis sind in den Zeilen 1 und 2 aufgeführt In Zeile 3 werden die Universalquantoren aus Zeile 2 eliminiert, und in Zeile 4 lautet die Behauptung abgeleitet mit modus ponens.

Der Kalkül bestehend aus den beiden angegebenen Inferenzregeln ist nicht vollständig. Es kann jedoch durch Hinzufügung weiterer Schlussfolgerungen zu einem vollständigen Verfahren erweitert werden Regeln. Diese nicht triviale Tatsache ist für Mathematik und KI von grundlegender Bedeutung. Der österreichische Logiker Kurt Gödel bewies 1931, dass [Göd31a]

Satz 3.3 (Gödels Vollständigkeitssatz) *Prädikatenlogik erster Ordnung ist vollständig. Das heißt, es gibt einen Kalkül, mit dem jeder Satz, der a ist Konsequenz einer Wissensbasis KB nachgewiesen werden kann. Wenn $KB \models \varphi$, dann gilt dass $KB \vdash \varphi$.*

Jeder wahre Satz in der Prädikatenlogik erster Ordnung ist daher beweisbar. Aber ist Gilt das umgekehrt auch? Ist alles, was wir syntaktisch ableiten können, tatsächlich wahr? Der Die Antwort lautet „Ja“:

Satz 3.4 (Korrektheit) *Es gibt Kalküle, mit denen nur wahre Aussagen bewiesen werden können. Das heißt, wenn $KB \vdash \varphi$ gilt, dann gilt $KB \models \varphi$.*



Abb. 3.3 Die universelle Logikmaschine

Tatsächlich sind fast alle bekannten Kalküle korrekt. Schließlich macht es wenig Sinn zu arbeiten mit falschen Beweismethoden. Beweisbarkeit und semantische Konsequenz sind also äquivalente Konzepte, sofern korrekte und vollständige Analysis verwendet wird. Damit Prädikatenlogik erster Ordnung wird zu einem leistungsstarken Werkzeug für Mathematik und KI. Der Die oben genannten Kalküle des natürlichen Abzugs sind für eine Automatisierung eher ungeeignet.

Nur Auflösungsrechnung, die 1965 eingeführt wurde und im Wesentlichen funktioniert
Nur eine einfache Inferenzregel ermöglichte die Konstruktion leistungsstarker automatisierter Orem-Beweiser, die später als Inferenzmaschinen für Expertensysteme eingesetzt wurden.

3.5 Auflösung

Tatsächlich löste die korrekte und vollständige Auflösungsrechnung in den 1970er Jahren eine logische Euphorie aus. Viele Wissenschaftler glaubten, dass man fast jede Aufgabe formulieren könne der Wissensdarstellung und Argumentation in PL1 und lösen Sie es dann mit einem automatisierten Beweiser. Prädikatenlogik, eine kraftvolle, ausdrucksstarke Sprache, zusammen mit a Die vollständige Beweisrechnung schien die universelle intelligente Maschine zur Darstellung von Wissen und zur Lösung vieler schwieriger Probleme zu sein (Abb. 3.3).

Wenn man eine Reihe von Axiomen (d. h. eine Wissensdatenbank) und eine Abfrage als Eingabe in eine solche Logikmaschine einspeist, sucht die Maschine nach einem Beweis und gibt ihn – denn einer existiert und wird gefunden – als Ausgabe zurück. Auf der Grundlage von Gödels Vollständigkeitssatz und der Arbeit von Herbrand wurde viel in die Mechanisierung der Logik investiert. Die Vision einer Maschine, die mit einer willkürlichen, nicht widersprüchlichen PL1-Wissensbasis jede echte Frage beweisen könnte, war sehr verlockend. Dementsprechend werden bisher viele Beweiskalküle für PL1 entwickelt und in Form von Theo-Rem-Beweisenden realisiert. Als Beispiel beschreiben wir hier den historisch wichtigen und weit verbreiteten Auflösungskalkül und zeigen seine Fähigkeiten. Der Grund für die Auswahl der Auflösung als Beispiel für eine Beweisrechnung in diesem Buch ist, wie bereits erwähnt, ihre historische und didaktische Bedeutung. Heutzutage ist die Auflösung nur eine von vielen Berechnungen, die in Hochleistungs-

Wir beginnen damit, den Beweis in Tabelle 3.2 auf Seite 41 mit der Wissensbasis von Beispiel 3.2 zu einem Auflösungsbeispiel zusammenzustellen. Zunächst werden die Formeln in die konjunktive Normalform und die negierte Abfrage umgewandelt

$$\neg Q \dot{\vee} \neg \text{kind}(\text{eve}, \text{oscar}, \text{anne})$$

wird zur Wissensdatenbank hinzugefügt, die Folgendes bietet

$$\begin{aligned} KB \dot{\vee} \neg Q \dot{\vee} (\text{Kind}(\text{Eve}, \text{Anne}, \text{Oscar}))1 \dot{\vee} \\ (\neg \text{Kind}(x, y, z) \dot{\vee} \text{Kind}(x, z, y))2 \dot{\vee} \\ (\neg \text{kind}(\text{eve}, \text{oscar}, \text{anne}))3. \end{aligned}$$

Der Beweis könnte dann etwa so aussehen

$$\begin{aligned} (2) \ x/\text{eve}, y/\text{anne}, z/\text{oscar} : (\neg \text{child}(\text{eve}, \text{anne}, \text{oscar}) \dot{\vee} \\ \text{child}(\text{eve}, \text{oscar}, \text{anne}))4 \\ \text{Res}(3, 4) : (\neg \text{child}(\text{eve}, \text{anne}, \text{oscar}))5 \\ \text{Res}(1, 5) : ()6, \end{aligned}$$

wobei im ersten Schritt die Variablen x,y,z durch Konstanten ersetzt werden. Anschließend folgen zwei Auflösungsschritte unter Anwendung der allgemeinen Auflösungsregel aus (2.2), die unverändert aus der Aussagenlogik übernommen wurde.

Etwas komplexer sind die Sachverhalte im folgenden Beispiel. Wir gehen davon aus, dass *jeder seine eigene Mutter kennt* und fragen, ob *Henry* jemanden kennt. Mit dem Funktionssymbol „Mutter“ und dem Prädikat „weiß“ müssen wir einen Widerspruch ableiten

$$(\text{weiß}(x, \text{Mutter}(x)))1 \dot{\vee} (\neg \text{weiß}(\text{Henry}, y))2.$$

Durch die Ersetzung x/Henry , $y/\text{Mother}(\text{Henry})$ erhalten wir das widersprüchliche Satzpaar

$$(\text{weiß}(\text{Henry}, \text{Mutter}(\text{Henry})))1 \dot{\vee} (\neg \text{weiß}(\text{Henry}, \text{Mutter}(\text{Henry})))2.$$

Dieser Ersetzungsschritt wird als *Vereinheitlichung bezeichnet*. Die beiden Literale ergänzen sich, was bedeutet, dass sie bis auf ihre Zeichen gleich sind. Die leere Klausel ist jetzt mit einem Auflösungsschritt ableitbar, durch den gezeigt wurde, dass Henry dies tut jemanden kennen (seine Mutter). Wir definieren

Definition 3.7 Zwei Literale heißen *unifizierbar*, wenn eine Substitution γ vorliegt für alle Variablen, wodurch die Literale gleich sind. Ein solches γ wird *Unifikator genannt*. Ein Unifier wird als der allgemeinste Unifier (MGU) bezeichnet, wenn alle anderen Unifier dies können daraus durch Substitution von Variablen gewonnen.

Beispiel 3.6 Wir wollen die Literale $p(f(g(x)), y, z)$ und $p(u, u, f(u))$ vereinheitlichen. Mehrere Vereiner sind

$\gamma_1 :$	$y/f(g(x)), z/f(f(g(x))), \gamma_2 : x/h(v), y/f$	$u/f(g(x)),$
	$(g(h(v))), z/f(f(g(h(v))))$, $\gamma_3 : x/h(h(v)), y/f(g(h(h(v))))$, $z/f(f(g(h(h(v))))$	$u/f(g(h(v)))$
	$(v))))$, $u/f(g(h(h(v))))$	
$\gamma_4 : x/h(a), y/f(g(h(a))), z/f(f(g(h(a))))$, $\gamma_5 : x/a, y/f(g(a)), z/f(f$	$u/f(g(h(a)))$	
$(g(a)))$,	$u/f(g(a))$	

wobei γ_1 der allgemeinste Unifikator ist. Die anderen Vereiner ergeben sich aus γ_1 durch Substitutionen $x/h(v)$, $x/h(h(v))$, $x/h(a)$, x/a .

An diesem Beispiel sehen wir, dass bei der Vereinheitlichung von Literalen die Prädikatssymbole wie Funktionssymbole behandelt werden können. Das heißt, das Literal wird wie ein Begriff implementierungen von Vereinheitlichungsalgorithmen verarbeiten die Argumente von Funktionen sequentiell. Begriffe werden über die Begriffsstruktur rekursiv vereinheitlicht. Die einfachsten Vereinheitlichungsalgorithmen sind in den meisten Fällen sehr schnell. Im schlimmsten Fall kann die Rechenzeit jedoch exponentiell mit der Größe der Terme wachsen. Denn für autom beweist, dass die überwiegende Zahl der Einigungsversuche scheitern oder sehr einfach sind. In den meisten Fällen hat die Worst-Case-Komplexität keine dramatischen Auswirkungen. Die schnellste Vereinigung Algorithmen haben selbst im schlimmsten Fall eine nahezu lineare Komplexität [Bib82].

Wir können nun die allgemeine Auflösungsregel für die Prädikatenlogik angeben:

Definition 3.8 Die Auflösungsregel für zwei Klauseln in konjunktiver Normalform liest

$$\frac{(A_1 \gamma \dots \gamma A_m \gamma B), (\neg B \quad \gamma C_1 \gamma \dots \gamma C_n) \gamma (B) = \gamma (B \gamma)}{(\gamma (A_1) \gamma \dots \gamma (A_m) \gamma \gamma (C_1) \gamma \dots \gamma (C_n)) \gamma \gamma}, (3.6)$$

wobei γ die MGU von B und B ist γ .

Satz 3.5 Die Auflösungsregel ist korrekt. Das heißt, die Auflösung ist eine semantische Konsequenz der beiden Elternsätze.

Der Vollständigkeit halber benötigen wir jedoch noch eine kleine Ergänzung, wie in der gezeigt folgendes Beispiel.

Beispiel 3.7 Das berühmte Russell-Paradoxon lautet: „Es gibt einen Friseur, der jeden rasiert, der sich nicht selbst rasiert.“ Diese Aussage ist widersprüchlich, also unbefriedigend. Das wollen wir mit Entschlossenheit zeigen. Das in PL1 formalisierte Paradoxon lautet:

$$\forall x \text{ Rasuren}(\text{Friseur}, x) \leftrightarrow \neg \text{Rasuren}(x, x)$$

und Transformation in Klauselform ergibt (siehe Übung 3.6 auf Seite 55)

$$(\neg \text{shaves}(\text{barbier}, x) \leftrightarrow \neg \text{shaves}(x, x))1 \leftrightarrow (\text{shaves}(\text{barbier}, x) \leftrightarrow \text{shaves}(x, x))2.$$

(3.7)

Aus diesen beiden Sätzen können wir mehrere Tautologien ableiten, aber keinen Widerspruch. Somit ist die Lösung nicht vollständig. Wir brauchen noch eine weitere Inferenzregel.

Definition 3.9 Die Faktorisierung einer Klausel erfolgt durch

$$\frac{(A1 \leftrightarrow A2 \leftrightarrow \dots \leftrightarrow A_n) \leftrightarrow (A1) \leftrightarrow (A2) \leftrightarrow (A2) \leftrightarrow \dots \leftrightarrow (A_n)}{\leftrightarrow (A_n)},$$

wobei $\leftrightarrow$ die MGU von A1 und A2 ist.

Nun lässt sich aus (3.7) ein Widerspruch ableiten

$$\begin{aligned} \text{Fak}(1, \leftrightarrow : x/\text{barber}) &: (\neg \text{shaves}(\text{barber}, \text{barber}))3 \\ \text{Fak}(2, \leftrightarrow : x/\text{Barber}) &: (\text{Rasuren}(\text{Barber}, \text{Barbier}))4 \\ \text{Res}(3, 4) &: ()5 \end{aligned}$$

und wir behaupten:

Satz 3.6 Die Auflösungsregel (3.6) ist zusammen mit der Faktorisierungsregel (3.9) vollständig widerlegt. Das heißt, durch Anwendung von Faktorisierungs- und Auflösungsschritten kann der leere Satz aus jeder unerfüllbaren Formel in konjunktiver Normalform abgeleitet werden.

3.5.1 Lösungsstrategien

Während die Vollständigkeit der Auflösung für den Benutzer wichtig ist, kann die Suche nach einem Beweis in der Praxis sehr frustrierend sein. Der Grund dafür ist der immense kombinatorische Suchraum. Auch wenn es zu Beginn nur sehr wenige Klauselpaare in $KB \dot{\vee} \neg Q$ gibt, generiert der Beweiser mit jedem Auflösungsschritt eine neue Klausel, wodurch sich die Anzahl der möglichen Auflösungsschritte in der nächsten Iteration erhöht. Daher wird seit langem versucht, den Suchraum durch spezielle Strategien möglichst ohne Verlust der Vollständigkeit zu reduzieren. Die wichtigsten Strategien sind die folgenden.

Bei der *Unit-Resolution* werden Auflösungsschritte priorisiert, bei denen eine der beiden Klauseln nur aus einem Literal besteht, einer sogenannten Unit-Klausel. Diese Strategie wahrt die Vollständigkeit und führt in vielen Fällen, aber nicht immer, zu einer Reduzierung des Suchraums. Es handelt sich also um einen heuristischen Prozess (siehe Abschn. 6.3).

Durch Anwendung der Set-of-Support-Strategie erhält man eine garantierte Reduzierung des Suchraums. Hier wird eine Teilmenge von $KB \dot{\vee} \neg Q$ als Supportmenge (SOS) definiert. Jeder Lösungsschritt muss eine Klausel aus dem SOS beinhalten, und die Lösung wird zum SOS hinzugefügt. Diese Strategie ist unvollständig. Sie wird vollständig, wenn sichergestellt ist, dass die Klauselmenge ohne das SOS erfüllbar ist (siehe Übung 3.7 auf Seite 55). Die negierte Abfrage $\neg Q$ wird häufig als anfängliches SOS verwendet.

Bei der *Eingabeauflösung* muss in jedem Auflösungsschritt eine Klausel aus der Eingabemenge $KB \dot{\vee} \neg Q$ beteiligt sein. Diese Strategie reduziert auch den Suchraum, allerdings auf Kosten der Vollständigkeit.

Mit der *reinen Literalregel* können alle Klauseln gelöscht werden, die Literale enthalten, zu denen es in anderen Klauseln keine komplementären Literale gibt. Diese Regel reduziert den Suchraum und ist vollständig und wird daher von praktisch allen Lösungsprüfern verwendet.

Wenn die Literale einer Klausel K_1 eine Teilmenge der Literale der Klausel K_2 darstellen, kann K_2 gelöscht werden. Zum Beispiel die Klausel

(regnet (heute) $\dot{\vee}$ street_wet (heute))

ist überflüssig, wenn *street_wet(today)* bereits gültig ist. Dieser wichtige Reduktionsschritt wird *Subsumtion* genannt. Auch die Subsumtion ist abgeschlossen.

3.5.2 Gleichheit

Gleichheit ist eine besonders unangenehme Ursache für das explosionsartige Wachstum des Suchraums. Wenn wir (3.1) auf Seite 36 und die in (3.2) auf Seite 36 formulierten Gleichheitsaxiome zur Wissensbasis hinzufügen, dann kann beispielsweise die Symmetrieklausel $\neg x = y \dot{\vee} y = x$ mit jeder positiven oder negierten Gleichung vereinheitlicht werden. Dies führt zur Ableitung neuer Klauseln und Gleichungen, auf die erneut Gleichheitsaxiome angewendet werden können, und so weiter. Die Transitivitäts- und Substitutionsaxiome haben ähnliche Konsequenzen. Aus diesem Grund wurden spezielle Gleichheitsschlussregeln entwickelt, die ohne explizite Gleichheitsaxiome auskommen und insbesondere den Suchraum reduzieren. *Demodulation*,

ermöglicht beispielsweise die Ersetzung von t_1 durch einen Term t_2 , wenn die Gleichung $t_1 = t_2$ existiert. Ein Gleichung $t_1 = t_2$ wird durch Vereinheitlichung auf einen Term t wie folgt angewendet:

$$\frac{t_1 = t_2, \quad (\dots t \dots), \quad \tilde{y}(t_1) = \tilde{y}(t)}{(\dots \tilde{y}(t_2) \dots)}.$$

Etwas allgemeiner ist die *Paramodulation*, die mit bedingten Gleichungen arbeitet [Bib82, Lov78].

Die Gleichung $t_1 = t_2$ erlaubt die Substitution des Termes t_1 durch t_2 sowie die Ersetzung von t_2 durch t_1 . Es ist in der Regel sinnlos, eine bereits erfolgte Ersetzung rückgängig zu machen durchgeführt worden. Im Gegenteil werden Gleichungen häufig zur Vereinfachung von Begriffen verwendet. Sie werden daher häufig nur in eine Richtung verwendet. Gleichungen, die nur in verwendet werden eine Richtung nennt man gerichtete Gleichungen. Eine effiziente Verarbeitung gerichteter Gleichungen wird durch sogenannte *Term-Rewriting-Systeme* erreicht. Für Formeln mit vielen Gleichungen gibt es spezielle Gleichheitsbeweiser.

3.6 Automatisierte Theorembeweiser

Implementierungen von Beweisrechnungen auf Computern werden als Theorembeweiser bezeichnet. Entlang mit spezialisierten Prüfern für Teilmengen von PL1 oder spezielle Anwendungen gibt es heute eine ganze Reihe automatisierter Beweiser für die vollständige Prädikatenlogik und Logik höherer Ordnung, von denen hier nur einige besprochen werden sollen. Das Wichtigste im Überblick Systeme finden Sie in [McC].

Einer der ältesten Auflösungsprüfer wurde im Oratorium des Argonne National Lab in Chicago entwickelt. Basierend auf frühen Entwicklungen ab 1963, Otter [Kal01], wurde 1984 gegründet. Otter wurde vor allem in Spezialgebieten erfolgreich eingesetzt der Mathematik, wie man auf seiner Homepage erfahren kann:

„Derzeit ist die Hauptanwendung von Otter die Forschung in abstrakter Algebra und formaler Logik. Otter und seine Vorgänger wurden zur Beantwortung vieler offener Fragen in den Bereichen verwendet endliche Halbgruppen, ternäre Boolesche Algebra, Logikkalküle, kombinatorische Logik, Gruppentheorie, Gittertheorie und algebraische Geometrie.“

Einige Jahre später entwickelte die Technische Universität München den Hochleistungsprüfer SETHEO [LSBB92] auf Basis der schnellen PROLOG-Technologie. Mit dem Ziel, eine noch höhere Leistung zu erreichen, wurde unter dem Namen PARTHEO eine Implementierung für Parallelrechner entwickelt. Es stellte sich heraus, dass es sich nicht lohnte, bei Theoremprüfern spezielle Hardware einzusetzen, wie dies auch in anderen Bereichen der Fall ist von KI, weil diese Computer sehr schnell von schnelleren Prozessoren überholt werden und intelligentere Algorithmen. München ist auch der Geburtsort von E [Sch02], einem preisgekrönten modernen Gleichungsbeweis, den wir im nächsten Kapitel kennenlernen werden Beispiel. Auf der Homepage von E ist folgende kompakte, ironische Charakterisierung zu lesen:

deren zweiter Teil übrigens für alle existierenden automatisierten Prüfer gilt
Heute.

„E ist ein reiner Gleichungssatzbeweiser für die Klausellogik. Das heißt, es ist ein Programm, das Sie können eine mathematische Spezifikation (in Klausellogik mit Gleichheit) und eine Hypothese hineinstopfen, die dann ewig läuft und alle Ressourcen Ihrer Maschine verbraucht. Sehr gelegentlich findet es einen Beweis für die Hypothese und sagt es Ihnen ;-).“

Beweise für wahre Aussagen zu finden ist offenbar so schwierig, dass die Suche nur äußerst selten oder – wenn überhaupt – erst nach sehr langer Zeit gelingt. Wir werden darauf eingehen dazu ausführlicher in Kap. 4. Hier sollte jedoch nicht nur erwähnt werden

Computern, aber auch die meisten Menschen haben Schwierigkeiten, strenge formale Beweise zu finden.

Obwohl es offensichtlich ist, dass Computer allein in vielen Fällen nicht in der Lage sind, einen Beweis zu finden, ist es die nächstbeste Lösung, Systeme zu bauen, die halbautomatisch arbeiten und ermöglichen eine enge Zusammenarbeit mit dem Benutzer. Dadurch kann der Mensch seine eigenen besser anwenden Kenntnisse über spezielle Anwendungsbereiche und schränken möglicherweise die Suche nach Beweisen ein. Einer der erfolgreichsten interaktiven Beweiser für Prädikatenlogik höherer Ordnung ist Isabelle [NPW02], ein gemeinsames Produkt der Universität Cambridge und der University of Cambridge Technik, München.

Wer auf der Suche nach einem leistungsstarken Prüfer ist, sollte einen Blick auf die aktuellen Ergebnisse des CASC (CADE ATP System Competition) [SS06] werfen.¹ Hier werden wir fündig der Gewinner von 2001 bis 2006 in den Kategorien PL1 und Clause Normalform war Manchesters Prüfer Vampire, der mit einer Auflösungsvariante und einem Special arbeitet Ansatz zur Gleichberechtigung. Das System Waldmeister des Max-Planck-Instituts in Saarbrücken ist seit Jahren führend in der Gleichstellungsprüfung.

Die vielen Spitzenplätze deutscher Systeme am CASC zeigen, dass die deutsche Forschung Gruppen im Bereich des automatisierten Theorembeweises spielen heute eine führende Rolle sowie in der Vergangenheit.

3.7 Mathematische Beispiele

Wir wollen nun die Anwendung eines automatisierten Beweisers mit dem oben genannten Beweiser E [Sch02] demonstrieren. E ist ein spezialisierter Gleichheitsprüfer, der stark schrumpft den Suchraum durch eine optimierte Gleichbehandlung.

Wir wollen beweisen, dass links- und rechtsneutrale Elemente in einer Halbgruppe gleich sind. Zunächst formalisieren wir den Anspruch Schritt für Schritt.

¹CADE ist die jährliche „Conference on Automated Deduction“ [CAD] und ATP steht für „Automated Theorem Prover“.

Definition 3.10 Eine Struktur $(M, \cdot)$, die aus einer Menge M mit einer zweistelligen inneren Operation „ $\cdot$ “ besteht, heißt Halbgruppe, wenn das Gesetz der Assoziativität gilt

$$\forall x \forall y \forall z (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

hält. Ein Element $e \in M$ heißt linksneutral (rechtsneutral), wenn $\forall x e \cdot x = x$ ($\forall x x \cdot e = x$).

Das muss noch gezeigt werden

Satz 3.7 Wenn eine Halbgruppe ein linksneutrales Element el und ein rechtsneutrales Element er hat, dann ist $el = er$.

Zuerst beweisen wir den Satz semiformal durch intuitives mathematisches Denken. Offenbar gilt für alle $x \in M$, dass

$$el \cdot x = x \quad (3.8)$$

Und

$$x \cdot er = x. \quad (3.9)$$

Wenn wir in (3.8) $x = er$ und in (3.9) $x = el$ setzen, erhalten wir die beiden Gleichungen $el \cdot er = er$ und $el \cdot er = el$. Die Verbindung dieser beiden Gleichungen ergibt

$$el = el \cdot er = er,$$

was wir beweisen wollen. Im letzten Schritt haben wir übrigens die Tatsache ausgenutzt, dass Gleichheit symmetrisch und transitiv ist.

Bevor wir den automatisierten Beweis anwenden, führen wir den Auflösungsnachweis manuell durch. Zunächst formalisieren wir die negierte Abfrage und die Wissensbasis KB , bestehend aus den Axiomen als Klauseln in konjunktiver Normalform:

$$\begin{aligned} (\neg el = er) & 1 & \text{negierte Abfrage} \\ (m(m(x, y), z) = m(x, m(y, z))) & 2 & \\ = x) & 3 & \\ (m(x, er) = & 4 & \end{aligned}$$

Gleichheitsaxiome:

$$\begin{aligned} (x = x) & 5 & \text{(Reflexivität)} \\ (\neg x = y \rightarrow y = x) & 6 & \text{(Symmetrie)} \\ (y \rightarrow \neg y = z \rightarrow x = z) & 7 & \text{(Transitivität)} \\ z) = m(y, z)) & 8 & \text{Substitution in } m \\ m(z, y)) & 9 & \text{Substitution in } m, \end{aligned}$$

wobei die Multiplikation durch das zweistellige Funktionssymbol m dargestellt wird. Die Gleichheitsaxiome wurden analog zu (3.1) auf Seite 36 und (3.2) auf Seite 36 formuliert. Ein einfacher Auflösungsbeweis hat die Form

$$\begin{aligned} &\text{Res}(3, 6, x6/m(e1, x3), y6/x3): \text{Res}(7, (x = m(e1, x))10 \\ &10, x7/x10, y7/m(e1, x10)): (\neg m(e1, x) = z \wedge x = z)11 \\ &\text{Res}(4, 11, x4/e1, x11/er, z11/e1): (er = e1)12 \\ &\text{Res}(1, 12, \tilde{y}): (). \end{aligned}$$

Hier bedeutet beispielsweise $\text{Res}(3, 6, x6/m(e1, x3), y6/x3)$ das in der Auflösung von Klausel 3 mit Klausel 6, die Variable x aus Klausel 6 wird durch $m(e1, x3)$ mit ersetzt Variable x aus Satz 3. Analog wird y aus Satz 6 durch x aus ersetzt Klausel 3.

Nun wollen wir den Beweiser E auf das Problem anwenden. Die Klauseln werden transformiert in die Klausel Normalformsprache LOP durch die Zuordnung

$$(\neg A1 \wedge \dots \wedge \neg Am \wedge B1 \wedge \dots \wedge Bn) \wedge B1; \dots; Bn < \tilde{y} A1, \dots, Am.$$

Die Syntax von LOP stellt eine Erweiterung der PROLOG-Syntax (siehe Abschn. 5) dar Nicht-Horn-Klauseln. Somit erhalten wir als Eingabedatei für E

$\leftarrow e1 = \tilde{a}h.$	% Abfrage
$m(m(X,Y),Z) = m(X,m(Y,Z)) . m(e1,X) = X$	% Assoziativität von m
$m(X,er) = X$	% linksneutrales Element von m
	% rechtsneutrales Element von m

wobei Gleichheit durch das Prädikatssymbol gl modelliert wird. Ein Anruf beim Prüfer liefert

```
unixprompt> eproof halbgr1a.lop
# Problemstatus ermittelt, Beweisobjekt erstellt
# Beweise für den Problemstatus beginnen
0 : [--equal(e1, er)] : initial
1 : [==equal(m(e1,X1), X1)] : initial
2 : [==equal(m(X1,er), X1)] : initial
3 : [==equal(e1, er)] : pm(2,1)
4 : [--equal(e1, e1)] : rw(0,3)
5 : [] : cn(4)
6 : [] : {Beweis}
# Der Beweis für den Problemstatus endet
```

Positive Literale werden durch $++$ und negative Literale durch $--$ gekennzeichnet. In den Zeilen 0 bis 4, Mit Initial markiert, werden die Klauseln aus den Eingabedaten noch einmal aufgelistet. $pm(a,b)$ steht für einen Auflösungsschritt zwischen Klausel a und Klausel b . Wir sehen darin den Beweis

Der von E gefundene Beweis ist dem manuell erstellten Beweis sehr ähnlich. Weil wir ausdrücklich Modellieren Sie die Gleichheit durch das Prädikat gl , kommen die besonderen Stärken von E nicht zum Tragen ins Spiel. Jetzt lassen wir die Gleichheitsaxiome weg und erhalten

$\leftarrow el = \hat{a}h.$	% Abfrage
$m(m(X,Y),Z) = m(X,m(Y,Z)) \cdot m(el,X) = X$	% Assoziativität v. m
$m(X,er) = X$	% linksneutrales El. v. m
	% rechtsneutrales El. v. m

als Eingabe für den Prüfer.

Der Beweis wird auch kompakter. Wir sehen in der folgenden Ausgabe des beweisen, dass der Beweis im Wesentlichen aus einem einzigen Inferenzschritt auf die beiden relevanten Klauseln 1 und 2 besteht.

```
unixprompt> eproof halbgr1a.lop
# Problemstatus ermittelt, Beweisobjekt erstellt
# Beweise für den Problemstatus beginnen
  0 : [¬equal(el, er)] : initial
  1 : [++equal(m(el,X1), X1)] : initial
  2 : [++equal(m(X1,er), X1)] : initial
  3 : [++equal(el, er)] : pm(2,1)
  4 : [¬equal(el, el)] : rw(0,3)
  5 : : cn(3)
  6 : [] : 5 : {Beweis}
# Der Beweis für den Problemstatus endet
```

Der Leser könnte nun einen genaueren Blick auf die Fähigkeiten von E werfen (Übung 3.9 weiter) . Seite 55).

3.8 Anwendungen

In der Mathematik werden automatisierte Theorembeweiser für bestimmte Spezialaufgaben eingesetzt.

Beispielsweise wurde der wichtige Vierfarbensatz der Graphentheorie erstmals bewiesen

1976 mit Hilfe eines Spezialprüfers. Automatisierte Prüfer spielen jedoch immer noch eine untergeordnete Rolle Rolle in der Mathematik.

Andererseits war zu Beginn der KI die Prädikatenlogik von großer Bedeutung für die Entwicklung von Expertensystemen in der Praxis. Aufgrund seiner Probleme Aufgrund der Modellierungsunsicherheit (siehe Abschn. 4.4) werden heute am häufigsten Expertensysteme entwickelt Verwendung anderer Formalismen.

Heutzutage spielt die Logik bei Verifikationsaufgaben eine immer wichtigere Rolle. Automatisch Programmverifikation ist derzeit ein wichtiges Forschungsgebiet zwischen KI und Software-Engineering. Heutzutage übernehmen immer komplexere Softwaresysteme Aufgaben

von immer größerer Verantwortungs- und Sicherheitsrelevanz. Hier ist ein Nachweis bestimmter Sicherheitseigenschaften eines Programms wünschenswert. Ein solcher Beweis kann nicht durch das Testen eines fertigen Programms erbracht werden, da es im Allgemeinen unmöglich ist, ein Programm auf alle möglichen Eingaben anzuwenden. Dies ist daher ein idealer Bereich für allgemeine oder sogar spezielle Inferenzsysteme. Unter anderem sind heute kryptografische Protokolle im Einsatz, deren Sicherheitseigenschaften automatisch überprüft wurden [FS97, Sch01]. Eine weitere Herausforderung für den Einsatz automatisierter Prüfer ist die Synthese von Software und Hardware. Zu diesem Zweck sollten Prüfer beispielsweise den Softwareentwickler bei der Generierung von Programmen aus Spezifikationen unterstützen.

Auch die Wiederverwendung von Software ist für viele Programmierer heutzutage von großer Bedeutung. Der Programmierer sucht nach einem Programm, das Eingabedaten mit bestimmten Eigenschaften entgegennimmt und ein Ergebnis mit gewünschten Eigenschaften berechnet. Ein Sortieralgorithmus akzeptiert Eingabedaten mit Einträgen eines bestimmten Datentyps und erstellt daraus eine Permutation dieser Einträge mit der Eigenschaft, dass jedes Element kleiner oder gleich dem vorherigen ist. Der Programmierer formuliert zunächst eine Spezifikation der Abfrage in PL1, bestehend aus zwei Teilen. Der erste Teil *PREQ* umfasst die Voraussetzungen, die erfüllt sein müssen, bevor das gewünschte Programm angewendet wird. Der zweite Teil *POSTQ* enthält die Nachbedingungen, die nach der Anwendung des gewünschten Programms gelten müssen.

Im nächsten Schritt muss eine Softwaredatenbank nach Modulen durchsucht werden, die diese Anforderungen erfüllen. Um dies formal zu überprüfen, muss die Datenbank für jedes Modul *M* eine formale Beschreibung der Vorbedingungen *PREM* und Nachbedingungen *POSTM* speichern. Eine Annahme über die Fähigkeiten der Module ist, dass sich die Vorbedingungen des Moduls aus den Vorbedingungen der Abfrage ergeben. Das muss es halten

$$PREQ \rightarrow PREM.$$

Alle Bedingungen, die als Voraussetzung für die Anwendung des Moduls *M* erforderlich sind, müssen als Vorbedingungen in der Abfrage erscheinen. Wenn beispielsweise ein Modul in der Datenbank nur Listen mit Ganzzahlen akzeptiert, müssen Listen mit Ganzzahlen als Eingabe auch als Vorbedingungen in der Abfrage erscheinen. Eine zusätzliche Anforderung in der Abfrage, dass beispielsweise nur gerade Zahlen vorkommen, stellt kein Problem dar.

Darüber hinaus muss für die Nachbedingungen gelten, dass

$$POSTM \rightarrow POSTQ.$$

Das heißt, nach Anwendung des Moduls müssen alle Attribute erfüllt sein, die die Abfrage erfordert. Wir zeigen nun die Anwendung eines Theorembeweisers auf diese Aufgabe an einem Beispiel aus [Sch01].

Beispiel 3.8 VDM-SL, die Vienna Development Method Specification Language, wird häufig als Sprache zur Spezifikation von Vor- und Nachbedingungen verwendet. Gehen Sie davon aus, dass in der Softwaredatenbank die Beschreibung eines Moduls *ROTATE* verfügbar ist, das das erste Listenelement an das Ende der Liste verschiebt. Wir suchen ein Modul *SHUFFLE*, das eine beliebige Permutation der Liste erstellt. Die beiden Spezifikationen lauten

ROTATE($l : \text{Liste}$) $\ddot{y} : \text{Liste}$

vorab *wahr*

Post

$(l = [] \ddot{y} l \quad \ddot{y} = []) \ddot{y}$

$(l = [] \ddot{y} l \quad \ddot{y} = (\text{Schwanz } l) \wedge [\text{Kopf } l])$

SHUFFLE($x : \text{Liste}$) $x \ddot{y} : \text{Liste}$

vorab *wahr*

post $\ddot{y} i : \text{Artikel}$

$(\ddot{y} x_1, x_2 : \text{Liste} \cdot x = x_1 \wedge [i] \wedge x_2 \ddot{y}$

$\ddot{y} y_1, y_2 : \text{Liste} \cdot x \quad \ddot{y} = y_1 \wedge [i] \wedge y_2)$

Hier steht für die Verkettung von Listen, und „ $\wedge$ “ trennt Quantoren mit ihre Variablen vom Rest der Formel. Die Funktionen „Kopf l “ und „Schwanz l “

Wählen Sie jeweils das erste Element und den Rest aus der Liste aus. Die Spezifikation von SHUFFLE gibt an, dass jedes Listenelement i , das vor dem in der Liste (x) war Die Anwendung von SHUFFLE muss im Ergebnis (x) enthalten sein nach der Bewerbung und umgekehrt umgekehrt. Es muss nun gezeigt werden, dass die Formel $(PREQ \ddot{y} PREM) \ddot{y} (POSTM \ddot{y} POSTQ)$ ist eine Folge der Wissensdatenbank, die eine Beschreibung der enthält Datentyp *Liste*. Die Beweisaufgabe ergibt sich aus den beiden VDM-SL-Spezifikationen

$\ddot{y} l, \ddot{y} x, x \ddot{y} : \text{Liste} \cdot (l = x \ddot{y} l \quad \ddot{y} \ddot{y} = x \quad \ddot{y} (w \ddot{y} w)) \ddot{y}$
 $(l = x \ddot{y} l \quad \ddot{y} = x \quad \ddot{y} ((l = [] \ddot{y} l \quad \ddot{y} = []) \ddot{y} (l = [] \ddot{y} l \quad \ddot{y} = (tl \ l) \wedge [hd \ l]))$
 $\ddot{y} \ddot{y} i : \text{Element} \cdot (\ddot{y} x_1, x_2 : \text{Liste} \cdot x = x_1 \wedge [i] \wedge x_2 \ddot{y} \ddot{y} y_1, y_2 : \text{Liste} \cdot x \quad \ddot{y} = y_1 \wedge [i] \wedge y_2)),$

was dann mit dem Beweiser SETHEO bewiesen werden kann.

In den kommenden Jahren dürfte das *Semantic Web* eine wichtige Anwendung von PL1 darstellen. Die Inhalte des World Wide Web sollen interpretierbar werden nicht nur für Menschen, sondern auch für Maschinen. Zu diesem Zweck werden Websites eingerichtet eine Beschreibung ihrer Semantik in einer formalen Beschreibungssprache. Die Suche nach Informationen im Web werden dadurch deutlich effektiver als heute, wobei im Wesentlichen nur Textbausteine durchsuchbar sind.

Als Beschreibungssprachen werden entscheidbare Teilmengen der Prädikatenlogik verwendet. Die Entwicklung effizienter Kalküle für das Denken ist sehr wichtig und eng damit verbunden zu den Beschreibungssprachen. Eine Abfrage für eine zukünftige semantisch arbeitende Suchmaschine könnte (informell) lauten: Wo in der Schweiz nächsten Sonntag in Höhen unter Wird es auf 2000 Metern gutes Wetter und bestens präparierte Skipisten geben? Um eine solche Frage zu beantworten, ist ein Kalkül erforderlich, der in der Lage ist, sehr schnell zu arbeiten auf großen Mengen von Fakten und Regeln. Hier sind komplexe verschachtelte Funktionsterme weniger wichtig.

Als grundlegendes Beschreibungsgerüst wurde das World Wide Web Consortium entwickelt die Sprache RDF (Resource Description Framework). Aufbauend auf RDF ermöglicht die deutlich leistungsfähigere Sprache OWL (Web Ontology Language) die Beschreibung von Beziehungen zwischen Objekten und Objektklassen, ähnlich wie PL1 [SET09]. *Ontologien* sind Beschreibungen von Beziehungen zwischen möglichen Objekten.

Eine Schwierigkeit beim Erstellen einer Beschreibung der unzähligen Websites ist der Arbeitsaufwand und auch die Überprüfung der Korrektheit der semantischen Beschreibungen. Hier können maschinelle Lernsysteme zur automatischen Generierung von Beschreibungen sehr hilfreich sein. Eine interessante Verwendung der „automatischen“ Generierung von Semantik im Web wurde von Luis von Ahn von der Carnegie Mellon University [vA06] vorgestellt. Er entwickelte Computerspiele, bei denen die über das Netzwerk verteilten Spieler gemeinsam Bilder mit Schlüsselwörtern beschreiben sollen. So wird den Bildern auf spielerische Art und Weise kostenlos eine Semantik zugewiesen. Der Leser kann die Spiele testen und einer Rede des Erfinders in Übung 3.10 auf Seite 55 [lauschen](#).

3.9 Zusammenfassung

Wir haben die wichtigsten Grundlagen, Begriffe und Verfahren der Prädikatenlogik bereitgestellt und gezeigt, dass selbst eine der schwierigsten intellektuellen Aufgaben, nämlich der Beweis mathematischer Theoreme, automatisiert werden kann. Automatisierte Beweiser können nicht nur in der Mathematik, sondern insbesondere bei Verifikationsaufgaben in der Informatik eingesetzt werden. Für das alltägliche Denken ist die Prädikatenlogik jedoch in den meisten Fällen ungeeignet. Im nächsten und den folgenden Kapiteln zeigen wir seine Schwachstellen und einige interessante moderne Alternativen. Darüber hinaus zeigen wir in Kap. 5, dass man mit Logik und ihren prozeduralen Erweiterungen elegant programmieren kann.

Wer sich für Logik erster Ordnung, Auflösung und andere Kalküle für automatisierte Beweiser interessiert, findet in [New00, Fit96, Bib82, Lov78, CL73] gute Anleitungen für Fortgeschrittene. Verweise auf Internetressourcen finden Sie auf der Website dieses Buches.

3.10 Übungen

Aufgabe 3.1 Gegeben seien das dreistellige Prädikat „Kind“ und das einstellige Prädikat „fe male“ aus Beispiel 3.2 auf Seite 35. Definieren Sie: (a)

Ein einstelliges Prädikat „männlich“.

(b) Ein zweistelliges Prädikat „Vater“ und „Mutter“. (c)

Ein zweistelliges Prädikat „Geschwister“.

(d) Ein Prädikat „Eltern(x, y, z)“, das genau dann wahr ist, wenn x der Vater und y ist die Mutter von z .

(e) Ein Prädikat „Onkel(x, y)“, das genau dann wahr ist, wenn x der Onkel von y ist (verwenden Sie die bereits definierten Prädikate).

(f) Ein zweistelliges Prädikat „Vorfahre“ mit der Bedeutung: *Vorfahren sind Eltern, Großeltern usw. beliebig vieler Generationen.*

Aufgabe 3.2 Formulieren Sie die folgenden Aussagen in der Prädikatenlogik:

(a) Jeder Mensch hat einen Vater und eine

Mutter. (b) Manche Menschen

haben Kinder. (c) Alle Vögel fliegen.

(d) Es gibt ein Tier, das (einige) getreidefressende Tiere frisst. (e) Jedes Tier frisst Pflanzen oder pflanzenfressende Tiere, die viel kleiner sind als selbst.

Übung 3.3 Passen Sie Übung 3.1 auf Seite 54 an, indem Sie einstellige Funktionssymbole und Gleichheit anstelle von „Vater“ und „Mutter“ verwenden.

Aufgabe 3.4 Geben Sie prädikatenlogische Axiome für die zweistellige Relation „<“ als Gesamtordnung an. Für eine Gesamtordnung müssen wir Folgendes haben: (1) Zwei beliebige Elemente sind vergleichbar. (2) Es ist symmetrisch. (3) Es ist transitiv.

Aufgabe 3.5 Vereinheitlichen Sie (wenn möglich) die folgenden Begriffe und geben Sie die MGU und die resultierenden

Begriffe an. (a) $p(x, f(y))$, $p(f(z),$

$u)$ (b) $p(x, f(x))$, $p(y, y)$ (c) x

$= 4 \cdot 7 \cdot x$, $\cos y = z$ (d) $x < 2 \cdot x$,

$3 < 6$ (e) $q(f(x, y, z), f$

$(g(w, w), g(x, x), g(y, y)))$, $q(u, u)$

Aufgabe 3.6 (a)

Transformieren Sie Russells Paradoxon aus Beispiel 3.7 auf Seite 45 in CNF. (b) Zeigen Sie, dass die leere Klausel nicht durch Auflösung ohne Faktorisierung aus (3.7) auf Seite 45 abgeleitet werden kann. Versuchen Sie, dies intuitiv zu verstehen.

Übung 3.7 (a)

Warum ist die Auflösung mit der Menge der Unterstützungsstrategie

unvollständig? (b) Begründen Sie (ohne zu beweisen), warum die Unterstützungsstrategie vollständig wird, wenn $(KB \cup \neg Q) \text{SOS}$ ist erfüllbar.

(c) Warum ist die Lösung mit der reinen Literalregel vollständig?

Übung 3.8 Formulieren und beweisen Sie mit Auflösung, dass in einer Halbgruppe mit mindestens zwei verschiedenen Elementen a, b , einem linksneutralen Element e und einem linksnullelement n diese beiden Elemente unterschiedlich sein müssen, das heißt, dass $n \neq e$. Verwenden Sie die Demodulation, die das Ersetzen von „Gleiches durch Gleiches“ ermöglicht.

Aufgabe 3.9 Besorgen Sie sich den Theorembeweiser E [Sch02] oder einen anderen Beweiser und beweisen Sie die folgenden Aussagen. Vergleichen Sie diese Beweise mit

denen im Text. (a) Die Behauptung aus Beispiel 2.3

auf Seite 24. (b) Russells Paradoxon aus Beispiel 3.7 auf Seite

45. (c) Die Behauptung aus Aufgabe 3.8.

Übung 3.10 Testen Sie die Spiele www.espgame.org und www.peekaboom.org, die dabei helfen, Bildern im Web Semantik zuzuordnen. Hören Sie sich den Vortrag über Human Computing von Luis von Ahn an unter: <http://video.google.de/videoplay?docid=-8246463980976635143>.

Grenzen der Logik

4

4.1 Das Suchraumproblem

Wie bereits an mehreren Stellen erwähnt, gibt es bei der Suche nach einem Beweis fast immer bei jedem Schritt viele (je nach Kalkül potenziell unendlich viele) Möglichkeiten zur Anwendung von Folgerungsregeln. Das Ergebnis ist das bereits erwähnte explosionsartige Wachstum des Suchraums (Abb. 4.1 auf Seite 58). Im schlimmsten Fall müssen alle diese Möglichkeiten ausprobiert werden, um den Beweis zu finden, was in der Regel nicht in angemessener Zeit möglich ist.

Wenn wir automatisierte Beweiser oder Inferenzsysteme mit Mathematikern oder menschlichen Experten vergleichen, die Erfahrung in speziellen Bereichen haben, machen wir interessante Beobachtungen. Einerseits können erfahrene Mathematiker Theoreme beweisen, die für automatisierte Beweiser weit außerhalb der Reichweite liegen. Andererseits führen automatisierte Prüfer Zehntausende von Schlussfolgerungen pro Sekunde durch. Im Gegensatz dazu führt ein Mensch vielleicht eine Schlussfolgerung pro Sekunde durch. Obwohl menschliche Experten auf der Objektebene (also bei der Durchführung von Schlussfolgerungen) viel langsamer sind, lösen sie scheinbar schwierige Probleme viel schneller.

Dafür gibt es mehrere Gründe. Wir Menschen verwenden intuitive Kalküle, die auf einer höheren Ebene funktionieren und oft viele der einfachen Schlussfolgerungen eines automatisierten Beweisers in einem Schritt ausführen. Darüber hinaus verwenden wir Lemmata, also abgeleitete wahre Formeln, die wir bereits kennen und die wir daher nicht jedes Mal neu beweisen müssen. Mittlerweile gibt es auch maschinelle Prüfer, die mit solchen Methoden arbeiten. Aber selbst sie können noch nicht mit menschlichen Experten konkurrieren.

Ein weiterer, viel wichtigerer Vorteil von uns Menschen ist die Intuition, ohne die wir keine schwierigen Probleme lösen könnten [PS09]. Der Versuch, die Intuition zu formalisieren, verursacht Probleme. Erfahrungen in angewandten KI-Projekten zeigen, dass in komplexen Bereichen wie der Medizin (siehe Abschn. 7.3) oder der Mathematik die meisten Experten nicht in der Lage sind, dieses intuitive Metawissen verbal zu formulieren, geschweige denn zu formalisieren. Deshalb können wir dieses Wissen nicht programmieren oder in Form von *Heuristiken in Kalküle integrieren*. Heuristiken sind Methoden, die den Weg zum Ziel in vielen Fällen erheblich vereinfachen oder verkürzen, in manchen Fällen (meist selten) aber auch erheblich verlängern.



Abb. 4.1 Mögliche Folgen der Explosion eines Suchraums

zum Ziel. Die heuristische Suche ist nicht nur für die Logik, sondern allgemein für die Problemlösung in der KI wichtig und wird daher in Kap. ausführlich behandelt. 6.

Ein interessanter Ansatz, der seit etwa 1990 verfolgt wird, ist die Anwendung maschineller Lerntechniken auf das Erlernen von Heuristiken zur Steuerung der Suche nach Inferenzsystemen, die wir nun kurz skizzieren. Ein Auflösungsprüfer hat bei der Suche nach einem Beweis Hunderte oder mehr Möglichkeiten für Auflösungsschritte bei jedem Schritt, aber nur wenige führen zum Ziel. Ideal wäre es, wenn der Beweiser ein Orakel fragen könnte, welche zwei Klauseln er im nächsten Schritt verwenden soll, um den Beweis schnell zu finden. Es gibt Versuche, solche Proof-Directing-Module aufzubauen, die für den nächsten Schritt die verschiedenen Alternativen bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Falle einer Auflösung könnte die Bewertung der verfügbaren Klauseln durch eine Funktion berechnet werden, die einen Wert basierend auf der Anzahl positiver Literale, der Komplexität der Begriffe usw. für jedes Paar auflösbarer Klauseln berechnet.

Wie kann diese Funktion implementiert werden? Da dieses Wissen „intuitiv“ ist, ist der Programmierer damit nicht vertraut. Stattdessen versucht man, die Natur zu kopieren und nutzt maschinelle Lernalgorithmen, um aus erfolgreichen Beweisen zu lernen [ESS89, SE90]. Die Attribute aller Klauselpaare, die an erfolgreichen Auflösungsschritten beteiligt sind, werden als positiv gespeichert, und die Attribute aller nicht erfolgreichen Auflösungen werden als negativ gespeichert. Anschließend wird unter Verwendung dieser Trainingsdaten und eines maschinellen Lernsystems ein Programm generiert, das Satzpaare heuristisch bewerten kann (siehe Abschn. 9.5).

Ein anderer, erfolgreicherer Ansatz zur Verbesserung des mathematischen Denkens ist gefolgt von interaktiven Systemen, die unter der Kontrolle des Benutzers arbeiten. Hier einer könnte Computeralgebraprogramme wie Mathematica, Maple oder Maxima nennen, das automatisch schwierige symbolische mathematische Manipulationen durchführen kann. Die Suche nach dem Beweis bleibt jedoch völlig dem Menschen überlassen. Deutlich mehr Unterstützung beim Beweis bietet die oben erwähnte interaktive Beweiserin Isabelle [NPW02]. suchen. Derzeit gibt es mehrere Projekte, darunter Omega [SB04] und MKM,¹ für die Entwicklung von Systemen zur Unterstützung von Mathematikern bei Beweisen.

Zusammenfassend kann man sagen, dass aufgrund der Suchraumproblematik automatisiert wird Beweiser können heute nur relativ einfache Theoreme in speziellen Bereichen mit beweisen wenige Axiome.

4.2 Entscheidbarkeit und Unvollständigkeit

Die Prädikatenlogik erster Ordnung bietet ein leistungsfähiges Werkzeug zur Darstellung von Wissen und Argumentation. Wir wissen, dass es korrekte und vollständige Kalküle und Theo-Rem-Beweiser gibt. Beim Beweis eines Theorems, also einer wahren Aussage, ist ein solcher Beweiser sehr hilfreich hilfreich, da man der Vollständigkeit halber nach endlicher Zeit weiß, dass die Aussage ist wirklich wahr. Was ist, wenn die Aussage nicht wahr ist? Der Vollständigkeitssatz (Theorem 3.3 auf Seite 41) beantwortet diese Frage nicht.² Konkret gibt es keinen Prozess die jede Formel aus PL1 in endlicher Zeit beweisen oder widerlegen kann, denn sie gilt

Satz 4.1 *Die Menge der gültigen Formeln in der Prädikatenlogik erster Ordnung ist semi-entscheidbar.*

Dieser Satz impliziert, dass es Programme (Theorembeweiser) gibt, die bei gegebener a Wenn Sie eine wahre (gültige) Formel als Eingabe verwenden, bestimmen Sie deren Wahrheit in endlicher Zeit. Wenn die Formel nicht stimmt gültig, es kann jedoch vorkommen, dass der Prüfer nie anhält. (Mit dieser Frage kann sich der Leser in Übung 4.1 auf Seite 65 auseinandersetzen.) Aussagenlogik ist entscheidbar denn die Wahrheitstabellenmethode liefert alle Modelle einer Formel in endlicher Zeit. Tatsächlich ist die Prädikatenlogik mit Quantoren und verschachtelten Funktionssymbolen eine Sprache etwas zu mächtig, um entscheidbar zu sein.

Andererseits ist die Prädikatenlogik für viele Zwecke nicht leistungsfähig genug. Oft möchte man Aussagen über Mengen von Prädikaten oder Funktionen machen. Das funktioniert in PL1 nicht, da es nur Quantoren für Variablen kennt, nicht jedoch für Prädikate oder Funktionen.

Kurt Gödel zeigte kurz nach seinem Vollständigkeitssatz für PL1, dass die Vollständigkeit verloren geht, wenn wir PL1 auch nur minimal erweitern, um eine Logik höherer Ordnung zu konstruieren. Eine Logik erster Ordnung kann nur über Variablen quantifizieren. Eine Logik zweiter Ordnung kann dies auch

¹www.mathweb.org/mathweb/demo.html.

² Gerade dieser Fall ist in der Praxis besonders wichtig, denn wenn ich bereits weiß, dass eine Aussage wahr ist, Ich brauche keinen Prüfer mehr.

über Formeln erster Ordnung quantifizieren, und eine Logik dritter Ordnung kann über Formeln zweiter Ordnung quantifizieren. Selbst wenn man nur das Induktionsaxiom für die natürlichen Zahlen hinzufügt, ist die Logik unvollständig. Die Aussage „Wenn ein Prädikat $p(n)$ für n gilt, dann gilt auch $p(n + 1)$ “ oder

$$\forall n (p(n) \rightarrow p(n + 1))$$

ist ein Satz zweiter Ordnung, da er über ein Prädikat quantifiziert. Gödel bewies den folgenden Satz:

Satz 4.2 (Gödels Unvollständigkeitssatz) *Jedes Axiomensystem für die natürlichen Zahlen mit Addition und Multiplikation (Arithmetik) ist unvollständig. Das heißt, es gibt wahre Aussagen in der Arithmetik, die nicht beweisbar sind.*

Gödels Beweis arbeitet mit der sogenannten Gödelisierung, bei der jede arithmetische Formel als Zahl kodiert wird. Es erhält eine eindeutige Gödel-Zahl. Zur Formulierung des Satzes wird nun die Gödelisierung verwendet

$$F = \text{„Ich bin nicht beweisbar.“}$$

in der Sprache der Arithmetik. Diese Formel ist aus folgendem Grund wahr. Angenommen, F ist falsch. Dann können wir F beweisen und damit zeigen, dass F nicht beweisbar ist. Das ist ein Widerspruch. Somit ist F wahr und daher nicht beweisbar.

Der tiefere Hintergrund dieses Theorems besteht darin, dass mathematische Theorien (Axiomensysteme) und ganz allgemein Sprachen unvollständig werden, wenn die Sprache zu mächtig wird. Ein ähnliches Beispiel ist die Mengenlehre. Diese Sprache ist so mächtig, dass man damit Paradoxien formulieren kann. Dabei handelt es sich um Aussagen, die in sich widersprüchlich sind, wie etwa die Aussage, die wir bereits aus Beispiel 3.7 auf Seite 45 kennen, über die Friseure, die alle diejenigen rasieren, die sich nicht selbst rasieren (siehe Übung 4.2 auf Seite 65).³ Das Dilemma besteht darin, dass mit Sprachen die aussagekräftig genug sind, um Mathematik und interessante Anwendungen zu beschreiben, schmuggeln wir Widersprüche und Unvollständigkeiten durch die Hintertür. Dies bedeutet jedoch nicht, dass Logiken höherer Ordnung für formale Methoden völlig ungeeignet sind. Es gibt sicherlich formale Systeme sowie Beweiser für Logik

4.3 Der fliegende Pinguin

Anhand eines einfachen Beispiels demonstrieren wir ein grundlegendes Problem der Logik und mögliche Lösungsansätze. Angesichts der Aussagen

³Viele weitere logische Paradoxien finden sich in [Wie].



Abb. 4.2 Der fliegende Pinguin Tweety

1. Tweety ist ein Pinguin
2. Pinguine sind Vögel
3. Vögel können

fliegen Formalisiert in PL1 ergibt die Wissensdatenbank *KB* :

Pinguin(tweety)
Pinguin(x) $\dot{\vee}$ Vogel(x)
Vogel(x) $\dot{\vee}$ Fliege(x)

Daraus lässt sich (z. B. mit Auflösung) „*fly(tweety)*“ ableiten (Abb. 4.2).⁴ Offensichtlich ist die Formalisierung der Flugattribute von Pinguinen unzureichend. Wir versuchen es mit der Zusatzaussage , dass *Pinguine nicht fliegen können*

Pinguin(x) $\dot{\vee}$ $\neg$ fly(x)

Daraus lässt sich *$\neg$ fly(tweety)* ableiten. Aber *Fly(tweety)* ist immer noch wahr. Die Wissensbasis ist daher inkonsistent. Hier bemerken wir ein wichtiges Merkmal der Logik, nämlich die Monotonie. Obwohl wir ausdrücklich darauf hinweisen, dass Pinguine nicht fliegen können, lässt sich die Gegenseite dennoch ableiten.

Definition 4.1 Eine Logik heißt monoton, wenn für eine beliebige Wissensbasis *KB* und eine beliebige Formel $\dot{\vee}$ die Menge der aus *KB* ableitbaren Formeln eine Teilmenge der aus *KB* ableitbaren Formeln $\dot{\vee}$ ist.

⁴Die formale Ausführung dieses und des folgenden einfachen Beweises kann dem Leser überlassen werden (Übung 4.3 auf Seite 65).

Wird ein Formelsatz erweitert, so können nach der Erweiterung weiterhin alle bisher ableitbaren Aussagen bewiesen werden, ggf. können auch weitere Aussagen bewiesen werden. Die Menge der beweisbaren Aussagen wächst also monoton, wenn die Menge der Formeln erweitert wird. Für unser Beispiel bedeutet das, dass die Erweiterung der Wissensbasis nie zum Ziel führen wird. Wir modifizieren daher *KB*, indem wir die offensichtlich falsche Aussage „(alle) Vögel können fliegen“ durch die genauere Aussage „(alle) Vögel außer Pinguinen können fliegen“ ersetzen und erhalten als *KB2* die folgenden Klauseln:

Pinguin(tweety)
Pinguin(x) $\dot{\vee}$ Vogel(x)
Vogel(x) $\dot{\vee}$ $\neg$ Pinguin(x) $\dot{\vee}$ Fliege(x)
Pinguin(x) $\dot{\vee}$ $\neg$ Fliege(x)

Nun ist die Welt offenbar wieder in Ordnung. Wir können *$\neg$ fly(tweety)* ableiten, aber nicht *fly(tweety)*, denn dafür bräuchten wir *$\neg$ penguin(x)*, was allerdings nicht ableitbar ist. Solange es auf dieser Welt nur Pinguine gibt, herrscht Frieden. Jeder normale Vogel macht jedoch sofort Probleme. Wir möchten den Raben Abraxas (aus dem deutschen Buch „Die kleine Hexe“) hinzufügen und erhalten

Rabe(abraxas)
Rabe(x) $\dot{\vee}$ Vogel(x)
Pinguin(tweety)
Pinguin(x) $\dot{\vee}$ Vogel(x)
Vogel(x) $\dot{\vee}$ $\neg$ Pinguin(x) $\dot{\vee}$ Fliege(x)
Pinguin(x) $\dot{\vee}$ $\neg$ Fliege(x)

Über die Flugeigenschaften von Abraxas können wir nichts sagen, da wir vergessen haben zu formulieren, dass Raben keine Pinguine sind. Daher erweitern wir *KB3* auf *KB4*:

Rabe(abraxas)
Rabe(x) $\dot{\vee}$ Vogel(x)
Rabe(x) $\dot{\vee}$ $\neg$ Pinguin(x)
Pinguin(tweety)
Pinguin(x) $\dot{\vee}$ Vogel(x)
Vogel(x) $\dot{\vee}$ $\neg$ Pinguin(x) $\dot{\vee}$ Fliege(x)
Pinguin(x) $\dot{\vee}$ $\neg$ fly(x)

Dass Raben keine Pinguine sind, was für den Menschen selbstverständlich ist, muss hier explizit hinzugefügt werden. Für den Aufbau einer Wissensbasis mit allen rund 9.800 Vogelarten weltweit muss daher für jede Vogelart (außer Pinguine) angegeben werden, dass sie nicht zu den Pinguinen gehört. Für alle anderen Ausnahmen wie den Strauß müssen wir analog vorgehen.

Für jedes Objekt in der Wissensdatenbank werden zusätzlich zu seinen Attributen alle Attribute, die es nicht hat, müssen aufgelistet werden.

Um dieses Problem zu lösen, wurden verschiedene Formen nichtmonotoner Logik entwickelt, die es ermöglichen, Wissen (Formeln) aus der Wissensbasis zu entfernen.

Unter dem Namen *Default-Logik* wurden Logiken entwickelt, die es ermöglichen, Objekten Attribute zuzuweisen, die gültig sind, solange keine anderen Regeln verfügbar sind. Im Tweety-Beispiel wäre die Regel „Vögel *können fliegen*“ eine solche *Standardregel*. Trotz großer Bemühungen haben sich diese Logiken aufgrund semantischer und praktischer Probleme derzeit nicht durchgesetzt.

Monotonie kann besonders bei komplexen Planungsproblemen, bei denen sich die Welt verändern kann, unangenehm sein. Wenn zum Beispiel ein blaues Haus rot gestrichen wird, dann ist es danach rot. Eine Wissensdatenbank wie z

Farbe(Haus, Blau)
Farbe(Haus, Rot)
 $\text{Farbe}(x, y) \dot{\vee} \text{Farbe}(x, y)$

lässt den Schluss zu, dass das Haus nach dem Anstrich rot und blau ist. Das hier bei der Planung auftretende Problem wird als *Rahmenproblem bezeichnet*. Eine Lösung hierfür ist die in Abschn. 2.1 vorgestellte Situationsrechnung. 5.6.

Ein interessanter Ansatz zur Modellierung von Problemen wie dem Tweety-Beispiel ist die Wahrscheinlichkeitstheorie. Die Aussage „Alle Vögel können fliegen“ ist falsch. Eine Aussage wie etwa „Fast alle Vögel können fliegen“ ist richtig. Diese Aussage wird genauer, wenn wir eine Wahrscheinlichkeit für „Vögel können fliegen“ angeben. Dies führt zur *probabilistischen Logik*, die heute einen wichtigen Teilbereich der KI und ein wichtiges Werkzeug zur Modellierung von Unsicherheit darstellt (siehe Kap. 7).

4.4 Modellierungsunsicherheit

Zweiwertige Logik kann und sollte nur Umstände modellieren, in denen es wahre, falsche und keine anderen Wahrheitswerte gibt. Für viele Aufgaben des alltäglichen Denkens ist die zweiwertige Logik daher nicht aussagekräftig genug. Die Regel

$\text{Vogel}(x) \dot{\vee} \text{Fliege}(x)$

gilt für fast alle Vögel, für einige ist sie jedoch falsch. Wie bereits erwähnt, ermöglicht die Arbeit mit Wahrscheinlichkeiten eine genaue Formulierung der Unsicherheit. Die Aussage „99 % aller Vögel können fliegen“ kann durch den Ausdruck formalisiert werden

$P(\text{Vogel}(x) \dot{\vee} \text{Fliege}(x)) = 0,99.$

Im Kap. 7 werden wir sehen, dass es hier besser ist, mit bedingten Wahrscheinlichkeiten wie zu arbeiten

$P(\text{Fliege}|\text{Vogel}) = 0,99.$

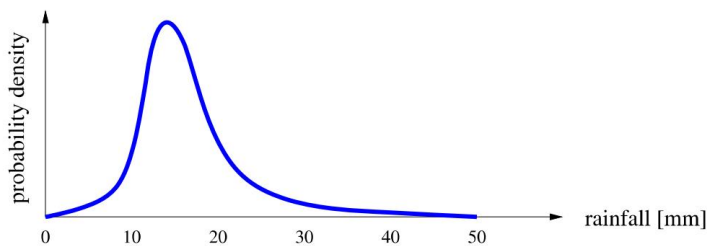


Abb. 4.3 Wahrscheinlichkeitsdichte des kontinuierlichen variablen Niederschlags

Mit Hilfe von *Bayes'schen Netzwerken* können komplexe Anwendungen mit vielen Variablen realisiert werden auch modelliert werden.

Für die Aussage „Das Wetter ist schön“ bedarf es eines anderen Modells . Hier ist es oft Es macht keinen Sinn, in Begriffen von wahr und falsch zu sprechen. Die Variable *Weather_is_nice* sollte nicht binär modelliert werden, sondern kontinuierlich mit Werten, zum Beispiel in das Intervall [0, 1]. *Weather_is_nice* = 0.7 bedeutet dann „Das Wetter ist ziemlich schön“. *Fuzzy-Logik*, die ebenfalls in Kap. vorgestellt wird. 7, wurde für diese Art von entwickelt kontinuierliche (Fuzzy-)Variable.

Auch die Wahrscheinlichkeitstheorie bietet die Möglichkeit, Aussagen über die Wahrscheinlichkeit stetiger Variablen zu treffen. Eine Aussage im Wetterbericht wie „Es gibt „Eine hohe Wahrscheinlichkeit, dass es etwas Regen geben wird“ könnte beispielsweise genau als Wahrscheinlichkeitsdichte der Form formuliert werden

$$P \text{ (Niederschlag} = X) = Y$$

und grafisch etwa wie in Abb. 4.3 dargestellt.

Diese sehr allgemeine und sogar visualisierbare Darstellung beider Arten von Unsicherheit, die wir diskutiert haben, ermöglicht es zusammen mit der induktiven Statistik und der Theorie der Bayes'schen Netzwerke im Prinzip, beliebige probabilistische Antworten zu geben Abfragen.

Wahrscheinlichkeitstheorie und Fuzzy-Logik sind nicht direkt mit Prädikaten vergleichbar Logik, weil sie keine Variablen oder Quantoren zulassen. Sie können daher als angesehen werden Erweiterungen der Aussagenlogik, wie in der folgenden Tabelle gezeigt.

Formalismus	Anzahl Wahrheitswerte	Wahrscheinlichkeiten ausdrückbar
Aussagelogik	2	–
Fuzzy-Logik	$\bar{y}$	–
Diskrete probabilistische Logik	N	Ja
Kontinuierliche probabilistische Logik	$\bar{y}$	Ja

4.5 Übungen

Übung 4.1

(a) Mit dem folgenden (falschen) Argument könnte man behaupten, dass PL1 entscheidbar ist:

Wir nehmen einen vollständigen Beweiskalkül für PL1. Damit können wir in endlicher Zeit einen Beweis für jede wahre Formel finden. Für jede andere Formel φ gehe ich wie folgt vor: Ich wende den Kalkül auf $\neg\varphi$ an und zeige, dass $\neg\varphi$ wahr ist. Somit ist φ falsch. Somit kann ich jede Formel in PL1 beweisen oder widerlegen. Finden Sie den Fehler im Argument und ändern Sie ihn, damit er

richtig wird. (b) Konstruieren Sie einen Entscheidungsprozess für die Menge der wahren und unerfüllbaren Formeln in PL1.

Übung 4.2 (a)

Gegeben sei die Aussage „Es gibt einen Friseur, der jeden rasiert, der sich nicht selbst rasiert.“

Überlegen Sie, ob sich dieser Friseur selbst rasiert. (b) Sei $M = \{x \mid x \text{ / } \varphi(x)\}$. Beschreiben Sie diese Menge und überlegen Sie, ob M sich selbst enthält.

Übung 4.3 Verwenden Sie einen automatisierten Theorembeweiser (z. B. E [Sch02]) und wenden Sie ihn auf alle fünf verschiedenen Axiomatisierungen des Tweety-Beispiels aus Abschn. 4.3 an.

4.3. Validieren Sie die Aussagen des Beispiels.

Logikprogrammierung mit PROLOG

5

Im Vergleich zu klassischen Programmiersprachen wie C oder Pascal schafft es Logik Es ist möglich, Beziehungen elegant, kompakt und deklarativ auszudrücken. Automatisiert Theorembeweiser sind sogar in der Lage zu entscheiden, ob eine Wissensbasis logisch ist bringt eine Abfrage mit sich. Beweisrechnung und Wissen werden in der Wissensdatenbank gespeichert streng getrennt. Als Eingabe kann eine im Abschnitt Normalform beschriebene Formel verwendet werden Daten für jeden Theorembeweiser, unabhängig von der verwendeten Beweisrechnung. Das ist großartig Wert für das Denken und die Darstellung von Wissen.

Will man Algorithmen implementieren, die zwangsläufig über prozedurale Komponenten verfügen, reicht eine rein deklarative Beschreibung oft nicht aus. Robert Kowalski, einer der Pioniere der Logikprogrammierung hat dies mit der Formel verdeutlicht

$$\text{Algorithmus} = \text{Logik} + \text{Kontrolle.}$$

Diese Idee wurde in der Sprache PROLOG verwirklicht. PROLOG wird verwendet in viele Projekte, vor allem in den Bereichen KI und Computerlinguistik. Wir geben jetzt eine Kurze Einführung in diese Sprache, Vorstellung der wichtigsten Konzepte, Darstellung der Sprache Stärken und vergleichen Sie es mit anderen Programmiersprachen und Theoremprüfern. Wer einen vollständigen Programmierkurs sucht, wird auf Lehrbücher wie z [Bra11, CM94] und die Handbücher [Wie04, Dia04].

Die Syntax der Sprache PROLOG erlaubt nur Horn-Klauseln. Logische Notation und die Syntax von PROLOG sind in der folgenden Tabelle gegenübergestellt:

PL1 / Klausel	Normalform	PROLOG	Beschreibung
$(\neg A_1 \dot{\vee} \dots \dot{\vee} \neg A_m \dot{\vee} B)$	$B :- A_1, \dots, A_m.$	Regel	
$(A_1 \dot{\vee} \dots \dot{\vee} A_m)$	B	$B :- A_1, \dots, A_m.$	Regel
A	$A.$		Tatsache
$(\neg A_1 \dot{\vee} \dots \dot{\vee} \neg A_m)$	$?$	$A_1, \dots, A_m.$	Abfrage
$\neg(A_1 \dot{\vee} \dots \dot{\vee} B \text{ in})$	$?$	$A_1, \dots, A_m.$	Abfrage

```

1 Kind (Oscar, Karen, Frank). 2 Kinder
(Mary, Karen, Frank). 3 Kinder (Eve,
Anne, Oscar). 4 Kinder (Henry,
Anne, Oscar). 5 Kind (Isolde, Anne,
Oscar). 6 Kinder (Clyde, Mary, Oscarb).
7 8 Kind(X,Z,Y) :- Kind(X,Y,Z). 9

10 Nachkomme(X,Y) :- Kind(X,Y,Z). 11
Nachkomme(X,Y) :- Kind(X,U,V), Nachkomme(U,Y).

```

Abb. 5.1 PROLOG-Programm mit Familienbeziehungen

Hier sind $A_1, \dots, A_m, A, B$ Literale. Die Literale werden wie in PL1 aus Prädikatssymbolen mit Termen als Argumenten konstruiert. Wie wir in der obigen Tabelle sehen können, gibt es in PROLOG keine Negationen im streng logischen Sinne, da das Vorzeichen eines Literals durch seine Position im Satz bestimmt wird.

5.1 PROLOG-Systeme und Implementierungen

Eine Übersicht über aktuelle PROLOG-Systeme finden Sie in der Linksammlung auf der Homepage dieses Buches. Dem Leser empfehlen wir die sehr leistungsfähigen und frei verfügbaren (unter GNU Public-Lizenzen) Systeme GNU-PROLOG [Dia04] und SWI-PROLOG. Für die folgenden Beispiele wurde SWI-PROLOG [Wie04] verwendet.

Die meisten modernen PROLOG-Systeme arbeiten mit einem Interpreter, der auf der *Warren Abstract Machine (WAM)* basiert. Der PROLOG-Quellcode wird in sogenannten WAM-Code kompiliert, der dann vom WAM interpretiert wird. Die schnellsten Implementierungen eines WAM verwalten bis zu 10 Millionen logische Schlussfolgerungen pro Sekunde (LIPS) auf einem 1-GH

5.2 Einfache Beispiele

Wir beginnen mit den Familienbeziehungen aus Beispiel 3.2 auf Seite 35. Die kleine Wissensbasis *KB* ist – ohne die Fakten für das Prädikat weiblich – als PROLOG-Programm mit dem Namen *rel.pl* in Abb. 5.1 codiert.

Mit dem Befehl kann das Programm im PROLOG-Interpreter geladen und übersetzt werden

?- [rel].

Eine erste Abfrage gibt den Dialog zurück

?- Kind (Eve, Oscar, Anne).

Ja

mit der richtigen Antwort Ja. Wie kommt es zu dieser Antwort? Für die Abfrage „?- child(eve,oscar,anne).“ Es gibt sechs Fakten und eine Regel mit demselben Prädikat im Satzkopf. Nun wird versucht, die Abfrage und jedes der komplementären Literale in den Eingabedaten in der Reihenfolge ihres Auftretens zu vereinheitlichen. Wenn eine der Alternativen fehlschlägt, wird zum letzten Verzweigungspunkt zurückverfolgt und die nächste Alternative getestet. Da die Vereinheitlichung bei jedem Fakt fehlschlägt, wird die Abfrage mit der rekursiven Regel in Zeile 8 vereinheitlicht. Nun versucht das System, das Unterziel child(eve,anne,oscar) Die Abfrage

?- Nachkomme(X,Y).

X = Oscar

Y = Karen

Ja

wird mit der ersten gefundenen Lösung beantwortet, so wie sie ist

?- Nachkomme(clyde,Y).

Y = Maria

Ja

Die Abfrage

?- Nachkomme(clyde,karen).

wird allerdings nicht beantwortet. Der Grund dafür ist die Klausel in Zeile 8, die die Symmetrie des untergeordneten Prädikats angibt. Diese Klausel ruft sich rekursiv auf, ohne dass eine Terminierung möglich ist. Dieses Problem kann mit dem folgenden neuen Programm gelöst werden (Fakten wurden hier weggelassen).

```
1 Nachkomme(X,Y) :- Kind(X,Y,Z). 2 Nachkomme(X,Y) :-
  Kind(X,Z,Y). 3 Nachkomme(X,Y) :- Kind(X,U,V),
  Nachkomme(U,Y).
```

Aber jetzt die Frage

?- Kind (Eve, Oscar, Anne).

wird nicht mehr richtig beantwortet, da die Symmetrie von child in den letzten beiden Variablen nicht mehr gegeben ist. Für beide Probleme findet sich im Programm eine Lösung

```

1 child_fact(oscar,karen,franz). 2
child_fact(mary,karen,franz). 3
child_fact(eva,anne,oscar). 4
child_fact(henry,anne,oscar). 5
child_fact(isolde,anne,oscar). 6
child_fact(clyde,mary,oscarb). 7 8 child(X,Z,Y) :-

child_fact(X,Y,Z). 9 child(X,Z,Y) :- child_fact(X,Z,Y).
10 11 Nachkomme(X,Y) :- Kind(X,Y,Z). 12

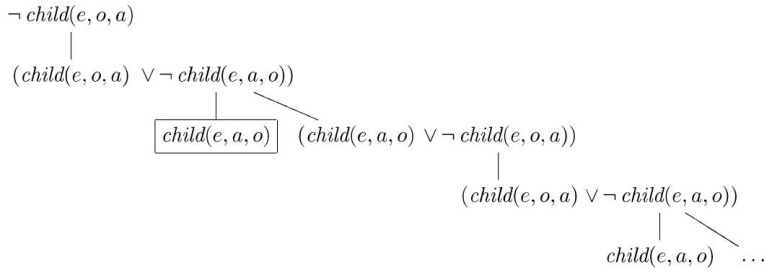
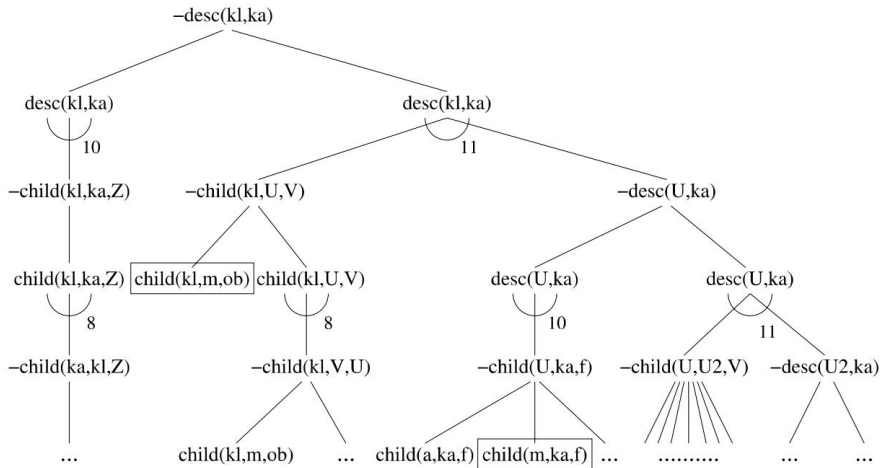
Nachkomme(X,Y) :- Kind(X,U,V), Nachkomme(U,Y).
```

Durch die Einführung des neuen Prädikats `child_fact` für die Fakten ist das Prädikat `child` nicht mehr rekursiv. Allerdings ist das Programm nicht mehr so elegant und einfach wie die – logisch korrekte – erste Variante in Abb. 5.1 auf Seite 68, die zur Endlosschleife führt. Der PROLOG-Programmierer muss, wie auch in anderen Sprachen, auf die Verarbeitung achten und Endlosschleifen vermeiden. PROLOG ist nur eine Programmiersprache und kein Theorembeweiser.

Wir müssen hier zwischen *deklarativer* und *prozeduraler Semantik* von PROLOG-Programmen unterscheiden. Die deklarative Semantik ist durch die logische Interpretation der Hornsätze gegeben. Die prozedurale Semantik hingegen wird durch die Ausführung des PROLOG-Programms definiert, die wir nun genauer betrachten wollen. Die Ausführung des Programms aus Abb. 5.1 auf Seite 68 mit der Abfrage `child(eve,oscar,anne)` ist in Abb. 5.2 auf Seite 71 als Suchbaum dargestellt.¹ Die Ausführung beginnt oben links mit der Abfrage. Jede Kante stellt einen möglichen SLD-Auflösungsschritt mit einem komplementären unfizierbaren Literal dar. Während der Suchbaum durch die rekursive Regel unendlich tief wird, wird die PROLOG-Ausführung beendet, da die Fakten vor der Regel in den Eingabedaten auftreten.

Bei der Abfrage „`descendent(clyde,karen)`“ hingegen wird die PROLOG-Ausführung nicht beendet. Wir können dies deutlich an dem in Abb. 5.3 auf Seite 71 dargestellten *Und-Oder-Baum* erkennen. In dieser Darstellung führen die Zweige, dargestellt durch, , vom Kopf eines Satzes zu den Unterzielen. Da alle Teilziele einer Klausel gelöst werden müssen, handelt es sich hierbei um *und-Zweige*. Alle anderen Zweige sind *oder-Zweige*, von denen mindestens einer mit seinen übergeordneten Knoten vereinbar sein muss. Die beiden skizzierte

¹Die Konstanten wurden aus Platzgründen abgekürzt.

Abb. 5.2 PROLOG-Suchbaum für `child(eve,oscar,anne)`Abb. 5.3 Und-Oder-Baum für `desc(clyde,karen)`

Lösung der Anfrage. Allerdings terminiert der PROLOG-Interpreter hier nicht, denn er arbeitet mit einer Tiefensuche mit Backtracking (siehe Abschn. 6.2.2) und wählt somit zunächst den unendlich tiefen Pfad ganz links.

5.3 Ausführungskontrolle und Verfahrenselemente

Wie wir am Beispiel einer Familienbeziehung gesehen haben, ist es wichtig, die Ausführung von PROLOG zu kontrollieren. Insbesondere die Vermeidung unnötiger Rückschritte kann zu erheblichen Effizienzsteigerungen führen. Ein Mittel hierzu ist der *Schnitt*. Indem wir ein Ausrufezeichen in eine Klausel einfügen, können wir verhindern, dass wir an diesem Punkt zurückgehen. Im folgenden Programm berechnet das Prädikat `max(X,Y,Max)` das Maximum der beiden Zahlen X und Y.


```
1 max(X,Y,X) :- X >= Y. 2 max(X,Y,Y) :-
  X < Y.
```

Wenn der erste Fall (erster Satz) zutrifft, wird der zweite nicht erreicht. Trifft der erste Fall hingegen nicht zu, dann ist die Bedingung im zweiten Fall wahr, was bedeutet, dass sie nicht überprüft werden muss. Zum Beispiel in der Abfrage

```
?- max(3,2,Z), Z > 10.
```

Backtracking wird eingesetzt, da $Z=3$ ist und die zweite Klausel auf max getestet wird, was zum Scheitern verurteilt ist. Daher ist ein Zurückverfolgen dieser Stelle nicht erforderlich. Dies können wir mit einem Schnitt optimieren:

```
1 max(X,Y,X) :- X >= Y, !. 2 max(X,Y,Y).
```

Daher wird die zweite Klausel nur dann aufgerufen, wenn sie wirklich notwendig ist, also wenn die erste Klausel fehlschlägt. Allerdings macht diese Optimierung das Programm schwerer verständlich.

Eine weitere Möglichkeit zur Ausführungskontrolle ist das eingebaute Prädikat fail, das niemals wahr ist. Im Beispiel der Familienbeziehung können wir mit der Abfrage ganz einfach alle Kinder und deren Eltern ausdrucken

```
?- child\_fact(X,Y,Z), write(X), write(' ist ein Kind von '), write(Y), write(' und '), write(Z),
  write('.') , nl, scheitern.
```

Die entsprechende Ausgabe ist

```
Oscar ist ein Kind von Karen und Frank. Mary ist ein Kind
von Karen und Frank. Eve ist ein Kind von Anne und
Oscar.
...
NEIN.
```

wobei das Prädikat nl einen Zeilenumbruch in der Ausgabe verursacht. Was wäre am Ende die Ausgabe ohne die Verwendung des Fail-Prädikats?

Mit derselben Wissensbasis wird die Abfrage „?- child_fact(ulla,X,Y)“ durchgeführt. würde zu der Antwort Nein führen, da es keine Fakten über Ulla gibt. Diese Antwort ist logisch nicht korrekt. Insbesondere lässt sich nicht beweisen, dass es kein Objekt mit dem Namen ulla gibt. Hier würde der Beweiser E korrekt antworten: „Kein Beweis gefunden.“ Wenn PROLOG also mit Nein antwortet, bedeutet dies nur, dass die Abfrage Q dies nicht kan

bewiesen werden. Dafür muss $\neg Q$ jedoch nicht unbedingt bewiesen werden. Dieses Verhalten wird als *Negation als Fehler bezeichnet*.

Die Beschränkung auf Horn-Klauseln stellt in den meisten Fällen kein großes Problem dar. Es ist jedoch wichtig für die Verfahrensausführung mittels SLD-Auflösung (Abschn. 2.5). Durch das einzeln ermittelte positive Literal pro Klausel haben die SLD-Auflösung und damit die Ausführung von PROLOG-Programmen einen eindeutigen Einstiegspunkt in die Klauseln. Nur so ist eine reproduzierbare Ausführung von Logikprogrammen und damit eine wohldefinierte prozedurale Semantik möglich.

Tatsächlich gibt es durchaus Problemstellungen, die nicht durch Horn-Klauseln beschrieben werden können. Ein Beispiel ist Russells Paradoxon aus Beispiel 3.7 auf Seite 45, das die Nicht-Horn-Klausel ($shaves(barber, X) \leftrightarrow shaves(X, X)$) enthält.

5.4 Listen

Als Hochsprache verfügt PROLOG wie die Sprache LISP über den praktischen generischen Listendatentyp. Eine Liste mit den Elementen A, 2, 2, B, 3, 4, 5 hat die Form

`[A,2,2,B,3,4,5]`

Das Konstrukt `[Head|Tail]` trennt das erste Element (Head) vom Rest (Tail) der Liste. Mit der Wissensdatenbank

`Liste([A,2,2,B,3,4,5]).`

PROLOG zeigt den Dialog an

`?- list([H|T]).`

`H=A`

`T = [2, 2, B, 3, 4, 5]`

Ja

Durch die Verwendung verschachtelter Listen können wir beliebige Baumstrukturen erstellen. Zum Beispiel die beiden Bäume



kann durch die Listen `[b,c]` bzw. `[a,b,c]` und die beiden Bäume dargestellt werden



durch die Listen $[[e,f,g],[h],d]$ bzw. $[a,[b,e,f,g],[c,h],d]$.

In den Bäumen, in denen die inneren Knoten Symbole enthalten, ist das Symbol der Kopf des Liste und die untergeordneten Knoten sind das Ende.

Ein schönes, elegantes Beispiel für die Listenverarbeitung ist die Definition des Prädikats `append(X,Y,Z)` zum Anhängen der Liste Y an die Liste X. Das Ergebnis wird in Z gespeichert. Die entsprechende PROLOG-Programme liest

```
1 anhängen([],L,L).
2 append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

Dies ist eine deklarative (rekursive) logische Beschreibung der Tatsache, dass L3 daraus resultiert Anhängen von L2 an L1. Gleichzeitig erledigt dieses Programm aber auch die Arbeit wenn es aufgerufen wird. Der Aufruf

```
?- append([a,b,c],[d,1,2],Z).
```

gibt die Substitution $Z = [a, b, c, d, 1, 2]$ zurück, genau wie der Aufruf

```
?- anhängen(X,[1,2,3],[4,5,6,1,2,3]).
```

ergibt die Substitution $X = [4, 5, 6]$. Hier stellen wir fest, dass `append` kein a ist Zwei-Stellen-Funktion, sondern eine Drei-Stellen-Beziehung. Eigentlich können wir auch das eingeben „Ausgabeparameter“ Z und fragen Sie, ob er erstellt werden kann.

Auch das Umkehren der Reihenfolge der Elemente einer Liste lässt sich elegant beschreiben und si durch das rekursive Prädikat gleichzeitig programmiert

```
1 nrev([],[]).
2 nrev([H|T],R) :- nrev(T,RT), append(RT,[H],R).
```

Dies reduziert die Umkehrung einer Liste auf die Umkehrung einer Liste, die aus einem Element besteht kleiner. Tatsächlich ist dieses Prädikat aufgrund des `Append`-Aufrufs sehr ineffizient. Dieses Programm ist als *Naive Reverse* bekannt und wird oft als PROLOG-Benchmark verwendet (siehe Übung 5.6 auf Seite 81). Die Dinge laufen besser, wenn man mit einem Provisorium fortfährt Speicher, der als Akkumulator bezeichnet wird, wie folgt:

Aufführen	Akkumulator
[A B C D] []	
[b,c,d]	[A]
[CD]	[b,a]
[D]	[c,b,a]
[]	[d,c,b,a]

Das entsprechende PROLOG-Programm liest

```
1 accrev([],A,A). 2
  accrev([H|T],A,R) :- accrev(T,[H|A],R).
```

5.5 Selbstmodifizierende Programme

PROLOG-Programme werden nicht vollständig kompiliert, sondern vom WAM interpretiert. Daher ist es möglich, Programme zur Laufzeit zu ändern. Ein Programm kann sich sogar selbst ändern. Mit Befehlen wie „Assert“ und „Retract“ können Fakten und Regeln zur Wissensbasis hinzugefügt oder daraus entnommen werden.

Eine einfache Anwendung der Variante Asserta ist das Hinzufügen abgeleiteter Fakten am Anfang der Wissensbasis mit dem Ziel, eine wiederholte, möglicherweise zeitaufwendige Ableitung zu vermeiden (siehe Übung 5.8 [auf](#) Seite 81). Wenn wir in unserem Familienbeziehungsbeispiel die beiden Regeln für den Prädikatsnachkommen durch ersetzen

```
1 :- dynamischer Nachkomme/2. 2
  Nachkomme(X,Y) :- Kind(X,Y,Z), Asserta(Nachkomme(X,Y)). 3 Nachkommen(X,Y) :- Kind(X,U,V),
  Nachkommen(U,Y), 4 Asserta(Nachkommen(X,Y)).
```

dann werden alle abgeleiteten Fakten für dieses Prädikat in der Wissensbasis gespeichert und somit in Zukunft nicht erneut abgeleitet. Die Abfrage

?- Nachkomme (Clyde, Karen).

führt zur Addition der beiden Tatsachen

Nachkomme (Clyde, Karen). Nachkomme
(Mary, Karen).

Durch die Manipulation von Regeln mit „Assert“ und „Retract“ können sogar Programme geschrieben werden, die sich selbst vollständig ändern. Diese Idee wurde unter dem Begriff *genetische Programmierung* bekannt. Es ermöglicht den Aufbau beliebig flexibler Lernprogramme. In der Praxis zeigt sich jedoch, dass eine Änderung des Codes durch Ausprobieren aufgrund der Vielzahl sinnloser Änderungsmöglichkeiten selten zu einer Leistungssteigerung führt. Die systematische Änderung von Regeln macht die Programmierung hingegen so viel komplexer, dass solche Programme, die ihren eigenen Code umfassend modifizieren, bisher keinen Erfolg hatten. Im Kap. 8 werden wir zeigen, wie maschinelles Lernen recht erfolgreich war. Allerdings werden hier nur sehr begrenzte Änderungen am Programmcode vorgenommen.

```

1 start :- action(state(left,left,left,left),
2               state(right,right,right,right)).
3
4 Aktionen (Start, Ziel): –
5   plan(Start,Ziel,[Start],Pfad),
6   nl,write("Lösung: "),nl,
7   write_path(Pfad).
8   % write_path(Path), fehlgeschlagen. 9           % aller Lösungen ausgegeben

10 Plan (Start, Ziel, Besucht, Pfad): –
11 go(Start>Weiter),
12 sicher>Weiter),
13 \+ member(Next,Visited), % not(member(...))
14   plan(Next,Goal,[Next|Visited],Path).
15 Plan(Ziel,Ziel,Pfad,Pfad).
16
17 go(state(X,X,Z,K),state(Y,Y,Z,K)):-across(X,Y). % Bauer, Wolf
18 go(state(X,W,X,K),state(Y,W,Y,K)):-across(X,Y). % Bauer, Ziege
19 go(state(X,W,Z,X),state(Y,W,Z,Y)):-across(X,Y). % Bauer, Kohl
20 go(state(X,W,Z,K),state(Y,W,Z,K)):-across(X,Y). % Bauer
21
22 quer (links, rechts).
23 quer (rechts, links).
24
25 sicher(Zustand(B,W,Z,K)):- über(W,Z), über(Z,K).
26 sicher(Zustand(B,B,B,K)).
27 sicher(Zustand(B,W,B,B)).

```

Abb. 5.4 PROLOG-Programm für das Landwirt-Wolf-Ziege-Kohl-Problem

5.6 Ein Planungsbeispiel

Beispiel 5.1 Das folgende Rätsel dient als Problemstellung für ein typisches PRO LOG-Programm.

Ein Bauer möchte einen Kohl, eine Ziege und einen Wolf über einen Fluss bringen, aber sein Boot ist so klein dass er sie nur einzeln überqueren kann. Der Bauer dachte darüber nach und sagte es dann selbst: „Wenn ich zuerst den Wolf auf die andere Seite bringe, dann frisst die Ziege den Kohl. Wenn ich Transportiere zuerst den Kohl, dann wird die Ziege vom Wolf gefressen. Was soll ich machen?“

Dies ist eine Planungsaufgabe, die wir mit etwas Überlegung schnell lösen können. Der Das in Abb. 5.4 dargestellte PROLOG-Programm wird nicht ganz so schnell erstellt.

Das Programm arbeitet mit Begriffen der Form state(Farmer,Wolf,Goat,Kohl), die den aktuellen Zustand der Welt beschreiben. Die vier Variablen mit mögliche Werte left, right geben den Standort der Objekte an. Der zentrale rekursive Prädikatplan erstellt zunächst einen Nachfolgezustand und testet anschließend mit go dessen Sicherheit mit sicher und wiederholt dies rekursiv, bis der Start- und Zielzustand gleich sind (in Programmzeile 15). Die bereits besuchten Staaten werden im gespeichert drittes Argument des Plans. Mit dem eingebauten Prädikat-Member wird geprüft, ob die state Next wurde bereits besucht. Wenn ja, wird es nicht versucht. Die Definition der Hier fehlt das Prädikat write_path für die Aufgabe, den gefundenen Plan auszugeben. Es wird als Übung für den Leser empfohlen (Übung 5.2 auf Seite 80). Für erste Pro-

Gramm testet das Literal `write_path(Path)`, das durch `write(Path)` ersetzt werden kann. Für die Abfrage „?- start.“ Wir bekommen die Antwort

Lösung:

Bauer und Ziege von links nach rechts
 Bauer von rechts nach links
 Bauer und Wolf von links nach rechts
 Bauer und Ziege von rechts nach links
 Bauer und Kohl von links nach rechts
 Bauer von rechts nach links
 Bauer und Ziege von links nach rechts

Ja

Zum besseren Verständnis beschreiben wir die Definition von `Plan` in der Logik:

$$\text{plan}(z, z) \text{ :- } \text{go}(s, n) \text{ :- } \text{sicher}(n) \text{ :- } \text{plan}(n, z) \text{ :- } \text{plan}(s, z)$$

Diese Definition fällt deutlich prägnanter aus als in PROLOG. Dafür gibt es zwei Gründe. Zum einen ist das Ergebnis des entdeckten Plans für die Logik unwichtig. Darüber hinaus ist es nicht wirklich notwendig zu prüfen, ob der nächste Staat bereits besucht wurde, wenn unnötige Fahrten den Landwirt nicht stören. Wenn jedoch `member(...)` im PROLOG-Programm weggelassen wird, liegt eine Endlosschleife vor und PROLOG findet möglicherweise keinen Zeitplan, selbst wenn einer vorhanden ist. Ursache hierfür ist die rückwärtsverkettende Suchstrategie von PROLOG, die nach dem Prinzip der Tiefensuche (Abschn. 6.2.2) immer auf Unterziele einzeln ohne Einschränkung der Rekursionstiefe arbeitet und daher unvollständig ist. Dies würde einem Theorembeweiser mit einem vollständigen Suchalgorithmus entsprechen.

Wie bei allen Planungsaufgaben ändert sich der Zustand der Welt, wenn von Schritt zu Schritt Aktionen ausgeführt werden. Dies legt nahe, den Zustand als Variable an alle Prädikate zu senden, die vom Zustand der Welt abhängen, beispielsweise im Prädikat `go`. Die Zustandsübergänge erfolgen im Prädikat `go`. Dieser Ansatz wird als *Situationskalkül* bezeichnet [RN10]. Eine interessante Erweiterung zum Erlernen von Aktionssequenzen in teilweise beobachtbaren, nichtdeterministischen Welten werden wir in Kapitel 7.3.2 sehen.

5.7 Constraint-Logik-Programmierung

Die Programmierung von Scheduling-Systemen, bei denen viele (zum Teil komplexe) logische und numerische Bedingungen erfüllt sein müssen, kann mit herkömmlichen Programmiersprachen sehr aufwändig und schwierig sein. Genau hier könnte Logik nützlich sein. Man schreibt einfach alle logischen Bedingungen in PL1 und gibt dann eine Abfrage ein. Normalerweise scheitert dieser Ansatz kläglich. Der Grund ist das in Abschn. 2.1 diskutierte Pinguinproblem. 4.3. Die Tatsache, dass Penguin(tweety) dafür sorgt, dass

WAHR. Es schließt jedoch nicht aus, dass raven(tweety) auch wahr ist. Dies mit zusätzlichen Axiomen auszuschließen ist sehr unpraktisch (Abschn. 4.3).

Constraint Logic Programming (CLP), das die explizite Formulierung von Einschränkungen für Variablen ermöglicht, bietet einen eleganten und sehr effizienten Mechanismus zur Lösung dieses Problems. Der Interpreter überwacht ständig die Ausführung des Programms auf Einhaltung aller Einschränkungen. Der Programmierer wird vollständig von der Kontrolle der Einschränkungen entlastet, was in vielen Fällen die Programmierung erheblich vereinfachen kann. Dies kommt im folgenden Zitat von Eugene C. Freuder aus [Fre97] zum Ausdruck:

Die Constraint-Programmierung stellt einen der bislang größten Ansätze der Informatik zum Heiligen Gral des Programmierens dar: Der Benutzer gibt das Problem vor, der Computer löst es.

Ohne auf die Theorie des Constraint Satisfaction Problem (CSP) einzugehen, wenden wir den CLP-Mechanismus von GNU-PROLOG auf das folgende Beispiel an.

Beispiel 5.2 Die Sekretärin des Albert-Einstein-Gymnasiums muss einen Plan für die Zuteilung von Räumen für Abschlussprüfungen ausarbeiten. Ihm liegen folgende Informationen vor: Die vier Lehrer Mayer, Hoover, Miller und Smith geben in den aufsteigend nummerierten Räumen 1, 2, 3 und 4 Tests für die Fächer Deutsch, Englisch, Mathematik und Physik. Jeder Lehrer gibt einen Test für genau einen ab Motiv in genau einem Raum. Darüber hinaus weiß er Folgendes über die Lehrer und ihre Fächer.

1. Herr Mayer testet nie in Raum 4.
2. Herr Miller testet immer Deutsch.
3. Herr Smith und Herr Miller führen keine Tests in benachbarten Räumen durch.
4. Frau Hoover testet Mathematik.
5. Physik wird immer im Raum Nummer 4 geprüft.
6. Deutsch und Englisch werden im Raum 1 nicht geprüft.

Wer führt in welchem Raum einen Test durch?

Ein GNU-PROLOG-Programm zur Lösung dieses Problems ist in Abb. 5.5 auf Seite 79 dargestellt. Dieses Programm arbeitet mit den Variablen Mayer, Hoover, Miller, Smith sowie German, English, Math, Physics, die jeweils eine ganze Zahl annehmen können. Geben Sie als Raumnummer einen Wert von 1 bis 4 ein (Programmzeilen 2 und 5). Eine Bindung Mayer = 1 und Deutsch = 1 bedeutet, dass Herr Mayer den Deutschtest in Raum 1 durchführt. Die Zeilen 3 und 6 sorgen dafür, dass die einzelnen vier Variablen unterschiedliche Werte annehmen. Zeile 8 sorgt dafür, dass im Lösungsfall allen Variablen ein konkreter Wert zugewiesen wird. Diese Zeile ist hier nicht zwingend erforderlich. Bei mehreren Lösungen würden jedoch nur Intervalle ausgegeben. In den Zeilen 10 bis 16 werden die Randbedingungen angegeben, in den restlichen Zeilen werden die Raumnummern für alle Lehrer und alle Fächer in einem einfachen Format ausgegeben.

Das Programm wird mit „[raumplan.pl].“ in GNU-PROLOG geladen, und mit „Start“. Wir erhalten die Ausgabe

[3,1,2,4]

[2,3,1,4]

```
1 Start:-
2 fd_domain([Mayer, Hoover, Miller, Smith],1,4),
3 fd_all_different([Mayer, Miller, Hoover, Smith]),
4
5 fd_domain([Deutsch, Englisch, Mathematik, Physik],1,4),
6 fd_all_different([Deutsch, Englisch, Mathematik, Physik]),
7
8 fd_labeling([Mayer, Hoover, Miller, Smith]),
9
10 Mayer #=4, 11 Miller           % Mayer nicht in Raum 4
   #= Deutsch, 12 dist(Miller,Smith) % Miller testet Deutsch
   #>= 2, 13 Hoover #= Mathematik, 14 Physik % Abstand Miller/Smith >= 2
   #= 4, 15 Deutsch #=1, 16           % Hoover testet Mathematik
   Englisch #=1, 17 nl,               % Physik in Raum 4
                                       % Deutsch nicht in Zimmer 1
                                       % Englisch nicht in Zimmer 1
18 write([Mayer, Hoover, Miller, Smith]), nl,
19 write([Deutsch, Englisch, Mathematik, Physik]), nl.
```

Abb. 5.5 CLP-Programm für das Raumplanungsproblem

Etwas praktischer dargestellt haben wir folgende Raumaufteilung:

Zimmernr.	1	2	3	4
Lehrer		Hoover Miller		Mayer Smith
Thema		Mathematik	Deutsch Englische	Physik

GNU-PROLOG verfügt, wie die meisten anderen CLP-Sprachen, über eine sogenannte *endliche Domäne Constraint-Löser*, mit dem Variablen ein endlicher Bereich ganzer Zahlen zugeordnet werden kann. Dabei muss es sich nicht zwangsläufig um ein Intervall wie im Beispiel handeln. Wir können auch eine Liste eingeben von Werten. Als Übung wird der Benutzer aufgefordert, in Übung 5.9 auf Seite 81 eine zu erstellen CLP-Programm, zum Beispiel mit GNU-PROLOG, für ein nicht ganz so einfaches Logikrätsel. Dieses angeblich von Einstein geschaffene Rätsel kann mit einem CLP ganz einfach gelöst werden System. Wenn wir dagegen versuchen würden, PROLOG ohne Einschränkungen zu verwenden, könnten wir es tun leicht die Zähne ausknirschen. Wer mit PROLOG bzw. eine elegante Lösung findet ein Prüfer, bitte lassen Sie es den Weg zum Autor finden.

5.8 Zusammenfassung

Vereinheitlichung, Listen, deklarative Programmierung und die relationale Sicht von Prozeduren, bei denen ein Argument eines Prädikats sowohl als Eingabe als auch als Ausgabe fungieren kann, ermöglichen die Entwicklung kurzer, eleganter Programme für viele Probleme. Viele Programme wären deutlich länger und daher schwieriger zu verstehen, wenn es in einer prozeduralen Form geschrieben wird Sprache. Darüber hinaus sparen diese Sprachfunktionen dem Programmierer Zeit. Daher ist PROLOG auch ein interessantes Werkzeug für Rapid Prototyping, insbesondere für KI

Anwendungen. Die CLP-Erweiterung von PROLOG ist nicht nur bei Logikrätseln hilfreich, sondern auch bei vielen Optimierungs- und Planungsaufgaben.

Seit seiner Erfindung im Jahr 1972 hat sich PROLOG in Europa neben prozeduralen Sprachen zu einer der führenden KI-Programmiersprachen Europas entwickelt. In den USA hingegen dominiert die einheimische Sprache LISP den KI-Markt.

PROLOG ist kein Theorembeweiser. Dies ist beabsichtigt, denn ein Programmierer muss die Verarbeitung einfach und flexibel steuern können und kommt mit einem Theorembeweis nicht weit. Andererseits ist PROLOG allein für den Beweis mathematischer Theoreme nicht sehr hilfreich. Allerdings gibt es durchaus interessante Theoretiker, die in PROLOG programmiert sind.

Als weiterführende Literatur werden [Bra11] und [CM94] sowie die empfohlenen Handbücher [Wie04, Dia04] und zum Thema CLP [Bar98].

5.9 Übungen

Aufgabe 5.1 Versuchen Sie, den Satz aus Abschn. zu beweisen. 3.7 über die Gleichheit links- und rechtsneutraler Elemente von Halbgruppen mit PROLOG. Welche Probleme treten auf? Was ist die Ursache dafür?

Aufgabe 5.2

- (a) Schreiben Sie ein Prädikat `write_move(+State1, +State2)`, das für jede Bootsüberquerung einen Satz wie „Bauer und Wolf kreuzen von links nach rechts“ ausgibt. `State1` und `State2` sind Begriffe der Form `state(Farmer, Wolf, Goat, Cabbage)`. (b)

Schreiben Sie ein rekursives Prädikat `write_path(+Path)`, das das Prädikat `write_move(+State1, +State2)` aufruft und alle Aktionen des Farmers ausgibt.

Aufgabe 5.3

- (a) Auf den ersten Blick ist die Variable `Path` im Prädikatplan des PROLOG-Programms aus Beispiel 5.1 auf Seite 76 unnötig, da sie offenbar nirgendwo geändert wird. Wofür wird es benötigt?
- (b) Wenn wir im Beispiel am Ende der Aktion ein Fail hinzufügen, werden alle Lösungen als Ausgabe ausgegeben. Warum wird jetzt jede Lösung zweimal gedruckt? Wie können Sie das verhindern?

Aufgabe 5.4

- (a) Zeigen Sie durch Testen, dass der Theorembeweiser E (im Gegensatz zu PROLOG) unter Berücksichtigung der Wissensbasis aus Abb. 5.1 auf Seite 68 die Abfrage „?- Deszendents(`clyde`, `karen`)“ beantwortet. korrekt. Warum das? (b)

Vergleichen Sie die Antworten von PROLOG und E für die Abfrage „?- Deszendents(`X`, `Y`)“.

Aufgabe 5.5 Schreiben Sie ein möglichst kurzes PROLOG-Programm, das 1024 Einsen ausgibt.

• **Übung 5.6** Untersuchen Sie das Laufzeitverhalten des naiven Umkehrprädikats.

- Führen Sie PROLOG mit der Trace-Option aus und beobachten Sie die rekursiven Aufrufe von `nrev`, `append` und `accrev`.
- Berechnen Sie die asymptotische Zeitkomplexität von `append(L1,L2,L3)`, also die Abhängigkeit der Laufzeit von der Länge der Liste für große Listen. Nehmen Sie an, dass der Zugriff auf den Kopf einer beliebigen Liste eine konstante Zeit benötigt.
- Berechnen Sie die Zeitkomplexität von `nrev(L,R)`.
- Berechnen Sie die zeitliche Komplexität von `accrev(L,R)`.
- Bestimmen Sie experimentell die Zeitkomplexität der Prädikate `nrev`, `append` und `accrev`, indem Sie beispielsweise Zeitmessungen durchführen (`Zeit(+Ziel)` liefert Rückschlüsse und CPU-Zeit).

Übung 5.7 Verwenden Sie Funktionssymbole anstelle von Listen, um die in Abschn. 5.7 angegebenen Bäume darzustellen. 5.4 auf Seite 73.

• **Übung 5.8** Die Fibonacci-Folge ist rekursiv definiert durch $\text{fib}(0) = 1$, $\text{fib}(1) = 1$ und $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$.

- Definieren Sie ein rekursives PROLOG-Prädikat `fib(N,R)`, das `fib(N)` und berechnet gibt es in `R` zurück.
- Bestimmen Sie die Laufzeitkomplexität des Prädikats `fib` theoretisch und anhand von Messung.
- Ändern Sie Ihr Programm durch die Verwendung von `Asserta so`, dass unnötige Schlussfolgerungen nicht mehr ausgeführt werden.
- Bestimmen Sie die Laufzeitkomplexität des geänderten Prädikats theoretisch und durch Messung (beachten Sie, dass dies davon abhängt, ob `fib` zuvor aufgerufen wurde).
- Warum ist `Fib` mit `Asserta` auch schneller, wenn es zum ersten Mal gestartet wird? nach dem Start von PROLOG?

• **Übung 5.9** Das folgende typische Logikrätsel wurde angeblich von Albert Einstein geschrieben. Darüber hinaus behauptete er angeblich, dass nur 2 % der Weltbevölkerung in der Lage seien, das Problem zu lösen. Folgende Aussagen werden gemacht.

- Es gibt fünf Häuser, jedes in einer anderen Farbe gestrichen.
- Jedes Haus wird von einer Person mit einer anderen Nationalität bewohnt.
- Jeder Bewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke und hat ein bestimmtes Haustier.
- Keiner der fünf Menschen trinkt das Gleiche, raucht das Gleiche oder hat das Gleiche dasselbe Haustier.

- Hinweise: – Der Brite wohnt im roten Haus.
- Der Schwede hat einen Hund.
- Der Däne trinkt gern Tee.
- Das grüne Haus befindet sich links vom weißen Haus.

- Der Besitzer des Gewächshauses trinkt Kaffee.
- Die Person, die Pall Mall raucht, hat einen Vogel.
- Der Mann, der im mittleren Haus wohnt, trinkt Milch.
- Der Besitzer des gelben Hauses raucht Dunhill.
- Der Norweger wohnt im ersten Haus.
- Der Marlboro-Raucher wohnt neben demjenigen, der eine Katze hat.
- Der Mann mit dem Pferd wohnt neben dem, der Dunhill raucht.
- Der Winfield-Raucher trinkt gerne Bier.
- Der Norweger wohnt neben dem blauen Haus.
- Der Deutsche raucht Rothmanns.
- Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt.

Frage: Wem gehört der Fisch? (a) Lösen Sie

das Rätsel zunächst manuell. (b)

Schreiben Sie ein CLP-Programm (zum Beispiel mit GNU-PROLOG), um das Rätsel zu lösen.

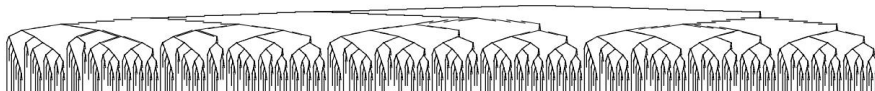
Orientieren Sie sich am Raumplanungsproblem in Abb. 5.5 auf Seite 79.

Suche, Spiele und Problemlösung

6

6.1 Einführung

Die Suche nach einer Lösung in einem extrem großen Suchbaum stellt für fast alle Inferenzsysteme ein Problem dar. Vom Ausgangszustand aus gibt es viele Möglichkeiten für den ersten Inferenzschritt. Für jede dieser Möglichkeiten gibt es im nächsten Schritt wiederum viele Möglichkeiten und so weiter. Selbst im Beweis einer sehr einfachen Formel aus [Ert93] mit drei Horn-Klauseln mit jeweils höchstens drei Literalen hat der Suchbaum für die SLD-Auflösung die folgende Form:



Der Baum wurde in einer Tiefe von 14 abgeschnitten und hat eine Lösung im mit γ markierten Blattknoten. Eine Darstellung ist nur aufgrund des geringen Verzweigungsfaktors von höchstens zwei und eines Cutoffs bei Tiefe 14 überhaupt möglich. Bei realistischen Problemen können Verzweigungsfaktor und Tiefe der ersten Lösung deutlich größer werden.

Angenommen, der Verzweigungsfaktor ist eine Konstante von 30 und die erste Lösung liegt in der Tiefe 50. Der Suchbaum hat $30^{50} \approx 7,4 \times 10^{73}$ Blattknoten. Die Anzahl der Inferenzschritte ist jedoch noch größer, da nicht nur jeder Blattknoten, sondern auch jeder innere Knoten des Baums einem Inferenzschritt entspricht. Daher müssen wir die Knoten über alle Ebenen addieren und so die Gesamtanzahl der Knoten des Suchbaums erhalten

$$\sum_{d=0}^{50} 30^d = \frac{1 - 30^{51}}{1 - 30} \approx \frac{30^{51}}{29} \approx 7,4 \times 10^{73}$$

was die Knotenanzahl nicht wesentlich ändert. Offensichtlich befinden sich fast alle Knoten dieses Suchbaums auf der letzten Ebene. Wie wir sehen werden, ist dies im Allgemeinen der Fall. Nun aber zurück zum Suchbaum mit den $7,4 \times 10^{73}$ Knoten. Nehmen wir an, wir hätten 10.000 Computer, von denen jeder eine Milliarde Schlussfolgerungen pro Sekunde durchführen kann, und das könnten wir

Verteilen Sie die Arbeit kostenlos auf alle Computer. Die Gesamtrechnenzeit für alle $7,4 \times 10^{73}$ -Schlussfolgerungen wäre ungefähr gleich

$$\frac{7,4 \times 10^{73}}{\times 109 \text{ Schlussfolgerungen/Sek}} = 7,4 \times 10^{60} \text{ Sek.} \approx 2,3 \times 10^{53} \text{ Jahre, } 10000$$

Das ist etwa 1043 Mal so viel Zeit wie das Alter unseres Universums. Durch diese einfache Gedankenübung können wir schnell erkennen, dass es keine realistische Chance gibt, einen solchen Suchraum mit den uns zur Verfügung stehenden Mitteln dieser Welt vollständig zu durchsuchen. Darüber hinaus waren die Annahmen zur Größe des Suchraums völlig realistisch. Beim Schach beispielsweise gibt es in einer typischen Situation über 30 mögliche Züge, und eine Partie mit 50 Halbrunden ist relativ kurz.

Wie kann es dann sein, dass es gute Schachspieler gibt – und heutzutage auch gute Schachcomputer? Wie kann es sein, dass Mathematiker Beweise für Theoreme finden, bei denen der Suchraum noch viel größer ist? Offensichtlich nutzen wir Menschen intelligente Strategien, die den Suchraum drastisch verkleinern. Der erfahrene Schachspieler wird ebenso wie der erfahrene Mathematiker durch bloße Beobachtung der Situation viele Handlungen sofort als sinnlos ausschließen. Aufgrund seiner Erfahrung ist er in der Lage, verschiedene Maßnahmen hinsichtlich ihres Nutzens für das Erreichen des Ziels zu bewerten. Oft geht man nach Gefühl vor. Fragt man einen Mathematiker, wie er einen Beweis gefunden hat, antwortet er möglicherweise, dass ihm die Intuition im Traum gekommen sei. In schwierigen Fällen stellen viele Ärzte die Diagnose rein nach dem Gefühl, basierend auf allen bekannten Symptomen. Gerade in schwierigen Situationen fehlt oft eine formale Lösungstheorie, die eine optimale Lösung garantiert. Bei alltäglichen Problemen, wie etwa der Suche nach einer entlaufenen Katze in Abb. 6.1 auf Seite 85, spielt die Intuition eine große Rolle. Mit dieser Art *heuristischer Suchmethode* beschäftigen wir uns in Abschn. 6.3 und beschreiben zusätzlich Prozesse, mit denen Com

Zunächst müssen wir jedoch verstehen, wie die *uninformierte Suche*, also das blinde Ausprobieren aller Möglichkeiten, funktioniert. Wir beginnen mit einigen Beispielen.

Beispiel 6.1 Mit dem 8-Puzzle, einem klassischen Beispiel für Suchalgorithmen [Nil98, RN10], lassen sich die verschiedenen Algorithmen sehr anschaulich veranschaulichen. Quadrate mit den Zahlen 1 bis 8 werden in einer 3×3 -Matrix wie in Abb. 6.2 auf Seite 86 verteilt. Ziel ist es, eine bestimmte Reihenfolge der Quadrate zu erreichen, zum Beispiel in aufsteigender Reihenfolge nach Zeilen, wie in Abb. dargestellt .6.2 auf Seite 86. In jedem Schritt kann ein Quadrat nach links, rechts, oben oder unten in den leeren Raum verschoben werden. Der leere Raum bewegt sich also in die entsprechende Gegenrichtung. Für die Analyse des Suchraums ist es sinnvoll, immer die möglichen Bewegungen des leeren Feldes zu betrachten.

Der Suchbaum für einen Startzustand ist in Abb. 6.3 auf Seite 86 dargestellt. Wir können sehen, dass der Verzweigungsfaktor zwischen zwei, drei und vier wechselt. Gemittelt über jeweils zwei Ebenen erhalten wir einen durchschnittlichen Verzweigungsfaktor¹ von $\approx 2,83$. Wir sehen, dass sich jeder Zustand zwei Ebenen tiefer mehrmals wiederholt, da bei einer einfachen uninformatierten Suche jede Aktion im nächsten Schritt rückgängig gemacht werden kann.

¹Der durchschnittliche Verzweigungsfaktor eines Baumes ist der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleicher Anzahl an Blattknoten hätte.



Abb. 6.1 Ein stark beschnittener Suchbaum – oder: „Wo ist meine Katze?“

Wenn wir Zyklen der Länge 2 nicht zulassen, erhalten wir für denselben Startzustand die Suchbaum dargestellt in Abb. 6.4 auf Seite 86. Der durchschnittliche Verzweigungsfaktor wird um etwa 1 reduziert und beträgt 1,8,2

2Bei einem 8-Puzzle hängt der durchschnittliche Verzweigungsfaktor vom Ausgangszustand ab (siehe Aufgabe 6.2) . Seite 110).

Abb. 6.2 Mögliche Ausgangs- und Zielzustände des 8er-Puzzles

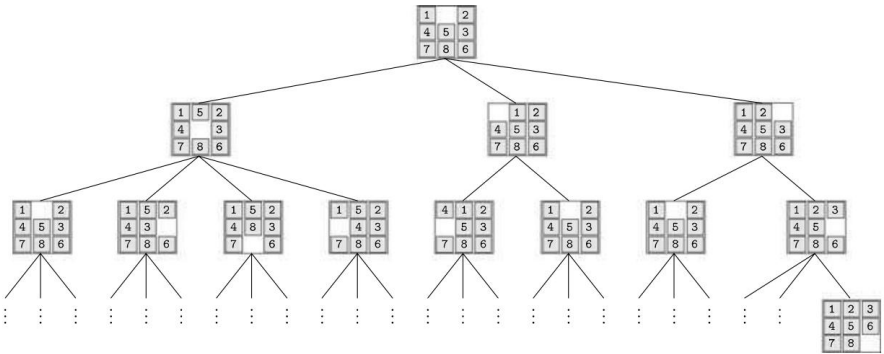


Abb. 6.3 Suchbaum für das 8-Puzzle. Unten rechts ist ein Zielzustand in Tiefe 3 dargestellt. Um Platz zu sparen, wurden die anderen Knoten auf dieser Ebene weggelassen

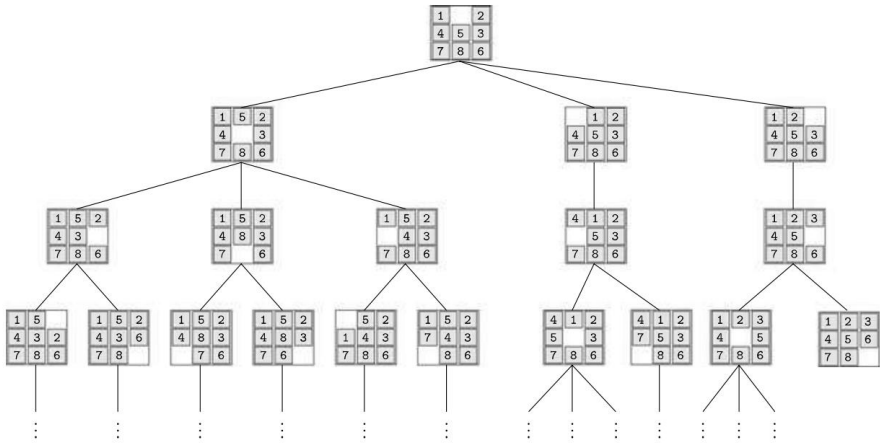


Abb. 6.4 Suchbaum für ein 8-Puzzle ohne Zyklen der Länge 2

Bevor wir mit der Beschreibung der Suchalgorithmen beginnen, sind einige neue Begriffe erforderlich. Wir haben es hier mit diskreten Suchproblemen zu tun. Im Zustand s ist eine Aktion $a_1 = a_1(s)$ einem neuen Zustand s' . Also $s' = a_1(s)$. Eine andere Aktion kann zu Zustand s führen, in führt zu andere Wörter: $s'' = a_2(s)$. Rekursive Anwendung aller möglichen Aktionen auf alle Zustände, Beginnend mit dem Startzustand ergibt sich der Suchbaum.

Definition 6.1 Ein Suchproblem wird durch die folgenden Werte definiert : *Zustand*: Beschreibung des Zustands der Welt, in der der Suchagent findet selbst.

Startzustand: Der Anfangszustand, in dem der Suchagent gestartet wird.

Zielzustand: Erreicht der Agent einen Zielzustand, beendet er sich und gibt eine Lösung aus (falls gewünscht).

Aktionen: Alle Agenten haben Aktionen zugelassen.

Lösung: Der Pfad im Suchbaum vom Startzustand zum Zielzustand.

Kostenfunktion: Weist jeder Aktion einen Kostenwert zu. Notwendig, um eine kostenoptimale Lösung zu finden.

Zustandsraum: Menge aller Zustände.

Auf das 8er-Puzzle angewendet erhalten wir

Zustand: 3x3-Matrix S mit den Werten 1, 2, 3, 4, 5, 6, 7, 8 (je einmal) und einer leeren Quadrat.

Ausgangszustand: Ein beliebiger Zustand.

Zielzustand: Ein beliebiger Zustand, z. B. der rechts in Abb. 6.2 auf Seite 86 angegebene Zustand.

Aktionen: Bewegung des leeren Quadrats S_{ij} nach links (wenn $j = 1$), nach rechts (wenn $j = 3$), nach oben (wenn $i = 1$), nach unten (wenn $i = 3$).

Kostenfunktion: Die konstante Funktion 1, da alle Aktionen gleiche Kosten haben.

Zustandsraum: Der Zustandsraum ist in Domänen degeneriert, die für beide Seiten nicht erreichbar sind (Übung 6.4 auf Seite 110). Somit gibt es unlösbare 8-Puzzle-Probleme.

Zur Analyse der Suchalgorithmen werden folgende Begriffe benötigt:

Definition 6.2 •

Die Anzahl der Nachfolgezustände eines Staates s wird als *Verzweigungsfaktor* bezeichnet $b(s)$ oder b , wenn der Verzweigungsfaktor konstant ist.

• Der *effektive Verzweigungsfaktor* eines Baums der Tiefe d mit insgesamt n Knoten ist definiert als der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleichem n hätte (siehe Übung 6.3 auf Seite 110). • Ein

Suchalgorithmus heißt *vollständig*, wenn er für jedes lösbare Problem eine Lösung findet.

Wenn ein vollständiger Suchalgorithmus abbricht, ohne eine Lösung zu finden, ist das Problem unlösbar.

Für eine gegebene Tiefe d und Knotenanzahl n kann der effektive Verzweigungsfaktor sein berechnet durch Lösen der Gleichung

$$n = \frac{b^{d+1} - 1}{b - 1} \quad (6.1)$$

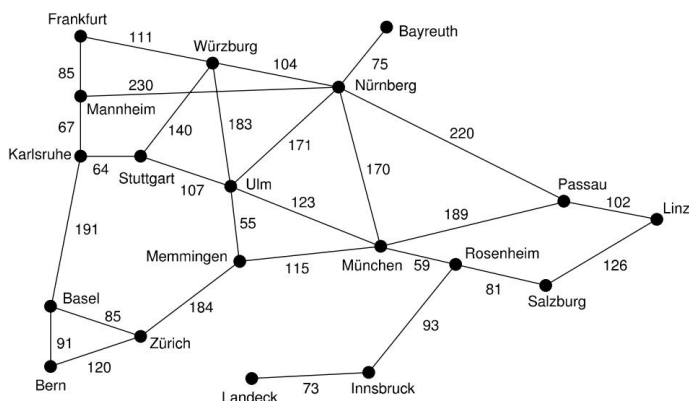


Abb. 6.5 Der Graph von Süddeutschland als Beispiel für eine Suchaufgabe mit Kostenfunktion

für b , weil ein Baum mit konstantem Verzweigungsfaktor und Tiefe d insgesamt hat

$$n = \sum_{i=0}^d b^i = \frac{b^{d+1} - 1}{b - 1} \quad (6.2)$$

Knoten.

Für die praktische Anwendung von Suchalgorithmen für endliche Suchbäume das Letzte Ebene ist besonders wichtig, weil

Satz 6.1 Bei stark verzweigten endlichen Suchbäumen mit einem großen konstanten Verzweigungsfaktor liegen fast alle Knoten auf der letzten Ebene.

Der einfache Beweis dieses Theorems wird dem Leser als Übung empfohlen (Übung 6.1 auf Seite 110).

Beispiel 6.2 Wir erhalten eine Karte, wie sie in Abb. 6.5 dargestellt ist, als Diagramm mit Städten als Knoten und Autobahnverbindungen zwischen den Städten als gewichtete Kanten mit Entfernungen. Wir suchen nach einer optimalen Route von Stadt A nach Stadt B. Die Beschreibung des entsprechenden Schemas lautet

Bundesland: Eine Stadt als aktueller Standort des Reisenden.

Ausgangszustand: Eine beliebige Stadt.

Zielzustand: Eine beliebige Stadt.

Aktionen: Reisen Sie von der aktuellen Stadt in eine Nachbarstadt.

Kostenfunktion: Die Entfernung zwischen den Städten. Jede Aktion entspricht einer Kante in der Grafik mit der Entfernung als Gewicht.

Zustandsraum: Alle Städte, also Knoten des Graphen.

Um die Route mit minimaler Länge zu finden, müssen die Kosten berücksichtigt werden, da diese nicht wie beim 8-Rätsel konstant sind.

Definition 6.3 Ein Suchalgorithmus heißt *optimal*, wenn er, sofern eine Lösung existiert, immer die Lösung mit den niedrigsten Kosten findet.

Das 8-Puzzle-Problem ist *deterministisch*, was bedeutet, dass jede Aktion von einem Zustand zu einem eindeutigen Nachfolgezustand führt. Darüber hinaus ist es *beobachtbar*, das heißt, der Agent weiß immer, in welchem Zustand er sich befindet. Bei der Routenplanung in realen Anwendungen sind beide Eigenschaften nicht immer gegeben. Die Aktion „Fahrt von München nach Ulm“ kann – beispielsweise aufgrund eines Unfalls – zum Nachfolgeland „München“ führen. Es kann auch vorkommen, dass der Reisende nicht mehr weiß, wo er ist, weil er sich verlaufen hat. Solche Komplikationen wollen wir zunächst einmal ignorieren. Daher werden wir in diesem Kapitel nur Probleme betrachten, die deterministisch und beobachtbar sind.

Probleme wie das 8-Puzzle, die deterministisch und beobachtbar sind, machen die Aktionsplanung relativ einfach, da es aufgrund eines abstrakten Modells möglich ist, Aktionssequenzen zur Lösung des Problems zu finden, ohne die Aktionen tatsächlich in der realen Welt auszuführen. Beim 8er-Rätsel ist es nicht notwendig, die Felder in der realen Welt tatsächlich zu verschieben, um die Lösung zu finden. Mit sogenannten *Offline-Algorithmen* können wir optimale Lösungen finden. Ganz andere Herausforderungen stellt man beispielsweise beim Bau von Robotern, die Fußball spielen sollen. Dabei wird es nie ein exaktes abstraktes Modell der Handlungen geben. Beispielsweise kann ein Roboter, der den Ball in eine bestimmte Richtung kickt, nicht mit Sicherheit vorhersagen, wohin sich der Ball bewegen wird, weil er unter anderem nicht weiß, ob ein Gegner den Ball fängt oder abwehrt. Hier sind dann *Online-Algorithmen* gefragt, die in jeder Situation Entscheidungen auf Basis von Sensorsignalen treffen. *Reinforcement Learning*, beschrieben in Abschn. 10 arbeitet daran, diese Entscheidungen auf der Grundlage von Erfahrungen zu optimieren.

6.2 Uninformierte Suche

6.2.1 Breitensuche

Bei der Breitensuche wird der Suchbaum von oben nach unten nach dem in Abb. 6.6 auf Seite 90 angegebenen Algorithmus durchsucht, bis eine Lösung gefunden ist. Zunächst wird jeder Knoten in der Knotenliste darauf getestet, ob es sich um einen Zielknoten handelt, und im Erfolgsfall wird das Programm gestoppt. Ansonsten werden alle Nachfolger des Knotens generiert. Anschließend wird die Suche rekursiv auf der Liste aller neu generierten Knoten fortgesetzt. Das Ganze wiederholt sich, bis keine Nachfolger mehr generiert werden.

Dieser Algorithmus ist generisch. Das heißt, es funktioniert für beliebige Anwendungen, wenn die beiden anwendungsspezifischen Funktionen „GoalReached“ und „Successors“ bereitgestellt werden. „GoalReached“ berechnet, ob das Argument ein Zielknoten ist, und „Successors“ berechnet die Liste aller Nachfolgerknoten seines Arguments. Abbildung 6.7 auf Seite 90 zeigt eine Momentaufnahme der Breitensuche.

```

BREADTH-FIRST-SEARCH(NodeList, Goal)
  Neue Knoten = ∅
  Für alle Node ∅ NodeList
    If GoalReached(Knoten, Ziel)
      Return(„Lösung gefunden“, Knoten)
    NewNodes = Append(NewNodes, Successors(Node))
  Wenn NewNodes = ∅
    Return(BREADTH-FIRST-SEARCH(NewNodes, Goal))
  Anders
    Return(„Keine Lösung“)

```

Abb. 6.6 Der Algorithmus für die Breitensuche

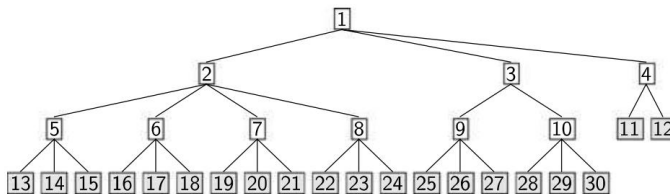


Abb. 6.7 Breitensuche während der Erweiterung der Knoten der dritten Ebene. Die Knoten werden entsprechend der Reihenfolge ihrer Generierung nummeriert. Die Nachfolger der Knoten 11 und 12 wurden noch nicht generiert

Analyse Da die Breitensuche jede Tiefe vollständig durchsucht und jede Tiefe in endlicher Zeit erreicht, ist sie vollständig, wenn der Verzweigungsfaktor b endlich ist.

Die optimale (also die kürzeste) Lösung liegt dann vor, wenn die Kosten aller Aktionen gleich sind (siehe Aufgabe 6.7 auf Seite 110). Rechenzeit und Speicherplatz wachsen exponentiell mit der Tiefe des Baums. Für einen Baum mit konstantem Verzweigungsfaktor b und Tiefe d ist die Gesamtzeit somit gegeben durch

$$C \cdot \sum_{i=0}^d b^i = \frac{b^{d+1} - 1}{b - 1} = O(b^d).$$

Obwohl nur die letzte Ebene im Speicher gespeichert wird, beträgt der Speicherplatzbedarf ebenfalls $O(b^d)$.

Bei der Geschwindigkeit heutiger Computer, die innerhalb von Minuten Milliarden von Knoten generieren können, ist der Hauptspeicher schnell voll und die Suche endet. Das Problem, dass nicht immer die kürzeste Lösung gefunden wird, kann durch die sogenannte *Uniform Cost Search* gelöst werden, bei der der Knoten mit den niedrigsten Kosten aufsteigend sortiert wird

```

DEPTH-FIRST-SEARCH (Knoten, Ziel)
If GoalReached(Node, Goal) Return("Lösung gefunden")
NewNodes = Nachfolger(Knoten)
Während NewNodes ≠ ∅
    Ergebnis = DEPTH-FIRST-SEARCH(First(NewNodes), Goal)
    Wenn Ergebnis = „Lösung gefunden“ Rückgabe („Lösung gefunden“)
    NewNodes = Rest(NewNodes)
Return("Keine Lösung")

```

Abb. 6.8 Der Algorithmus für die Tiefensuche. Die Funktion „First“ gibt das erste Element einer Liste zurück und „Rest“ den Rest der Liste

Die Liste der Knoten wird immer erweitert und die neuen Knoten einsortiert. So finden wir die optimale Lösung. Das Speicherproblem ist jedoch noch nicht gelöst. Eine Lösung für dieses Problem bietet die Tiefensuche.

6.2.2 Tiefensuche

Bei der Tiefensuche werden jeweils nur wenige Knoten gleichzeitig im Speicher gespeichert. Nach der Erweiterung eines Knotens werden nur dessen Nachfolger gespeichert und der erste Nachfolgerknoten wird sofort erweitert. Dadurch wird die Suche schnell sehr tiefgründig. Erst wenn ein Knoten keine Nachfolger hat und die Suche in dieser Tiefe fehlschlägt, wird der nächste offene Knoten durch *Backtracking* zum letzten Zweig usw. erweitert. Am besten erkennen wir dies am eleganten rekursiven Algorithmus in Abb. 6.8 und am Suchbaum in Abb. 6.9 auf Seite 92.

Analyse Die Tiefensuche erfordert viel weniger Speicher als die Breitensuche, da in jeder Tiefe höchstens b Knoten gespeichert werden. Wir benötigen also $b \cdot d$ Speicherzellen.

Allerdings ist die Tiefensuche für unendlich tiefe Bäume nicht vollständig, da die Tiefensuche in eine Endlosschleife gerät, wenn es im Zweig ganz links keine Lösung gibt. Daher ist die Frage nach der optimalen Lösung obsolet. Aufgrund der Endlosschleife kann keine Grenze für die Rechenzeit angegeben werden. Bei einem endlich tiefen Suchbaum mit der Tiefe d werden insgesamt etwa b Knoten erzeugt.

Somit wächst die Rechenzeit, genau wie bei der Breitensuche, exponentiell mit der Tiefe.

Wir können den Suchbaum endlich machen, indem wir eine Tiefenbegrenzung festlegen. Wenn nun im beschnittenen Suchbaum keine Lösung gefunden wird, kann es dennoch Lösungen außerhalb der Grenze geben. Dadurch wird die Suche unvollständig. Es gibt jedoch naheliegende Ideen, um die Suche zu vervollständigen.

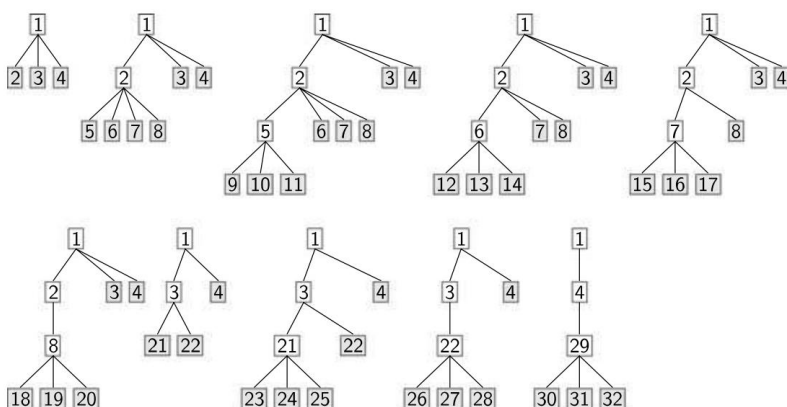


Abb. 6.9 Durchführung der Tiefensuche. Alle Knoten in Tiefe drei sind nicht erfolgreich und führen zu einem Backtracking. Die Knoten werden in der Reihenfolge nummeriert, in der sie generiert wurden

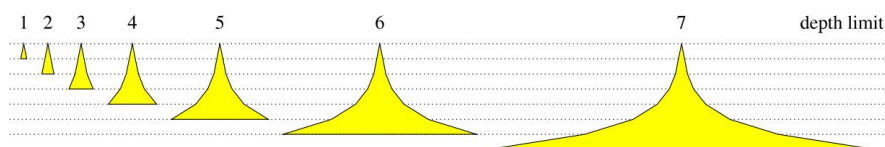


Abb. 6.10 Schematische Darstellung der Entwicklung des Suchbaums bei der iterativen Vertiefung mit Grenzen von 1 bis 7. Die Breite des Baums entspricht einem Verzweigungsfaktor von 2

6.2.3 Iterative Vertiefung

Wir beginnen die Tiefensuche mit einem Tiefenlimit von 1. Wenn keine Lösung gefunden wird, erhöhen wir das Limit um 1 und beginnen die Suche von vorne und so weiter, wie in Abb. 6.10 dargestellt. Diese iterative Erhöhung der Tiefengrenze wird als *iterative Vertiefung* bezeichnet.

Wir müssen das in Abb. 6.8 auf Seite 91 dargestellte Tiefensuchprogramm um die beiden zusätzlichen Parameter „Depth“ und „Limit“ erweitern. „Depth“ wird beim rekursiven Aufruf um eins erhöht und die Kopfzeile der while-Schleife wird durch „While NewNodes = $\neq$ And Depth < Limit“ ersetzt. Der modifizierte Algorithmus ist in Abb. 6.11 auf Seite 93 dargestellt.

Analyse Der Speicherbedarf ist derselbe wie bei der Tiefensuche. Man könnte argumentieren, dass das wiederholte Starten der Tiefensuche bei Tiefe Null eine Menge überflüssiger Arbeit verursacht. Bei großen Verzweigungsfaktoren ist dies nicht der Fall. Wir zeigen nun, dass die Summe der Anzahl der Knoten aller Tiefen bis zum vorletzten $d_{max} - 1$ in allen durchsuchten Bäumen viel kleiner ist als die Anzahl der Knoten im zuletzt durchsuchten

ITERATIVEDEEPENING(Knoten, Ziel)**DepthLimit = 0**

Wiederholen

Ergebnis = DEPTHFIRSTSEARCH-B(Knoten, Ziel, 0, DepthFirstSearch)**DepthLimit = DepthLimit + 1****Bis Ergebnis = „Lösung gefunden“****DEPTHFIRSTSEARCH-B(Knoten, Ziel, Tiefe, Limit)****If GoalReached(Node, Goal) Return(„Lösung gefunden“)****NewNodes = Nachfolger(Knoten)****Während NewNodes ≠ ∅ und Depth < Limit****Ergebnis =****DEPTHFIRSTSEARCH-B(First(NewNodes), Ziel, Tiefe + 1, Limit)****Wenn Ergebnis = „Lösung gefunden“ Rückgabe („Lösung gefunden“)****NewNodes = Rest(NewNodes)****Return(„Keine Lösung“)**

Abb. 6.11 Der Algorithmus zur iterativen Vertiefung, der die leicht modifizierte Tiefensuche mit Tiefenbegrenzung aufruft (TIEFENSUCHE-B)

Sei $N_b(d)$ die Anzahl der Knoten eines Suchbaums mit dem Verzweigungsfaktor b und Tiefe d und d_{max} sind die zuletzt gesuchte Tiefe. Der zuletzt durchsuchte Baum enthält

$$N_b(d_{max}) = \sum_{i=0}^{d_{max}} b^i = \frac{b^{d_{max}+1} - 1}{b - 1}$$

Knoten. Alle Bäume wurden vorher gemeinsam durchsucht

$$\begin{aligned}
 N_b(d) &= \sum_{d=1}^{d_{max}} \sum_{d=1}^{d_{max}-1} b^{d+1} = \frac{1}{b-1} \sum_{d=1}^{d_{max}} b^{d+1} \\
 &= \frac{1}{b-1} \sum_{d=2}^{d_{max}} b^d = \frac{1}{b-1} \frac{b^{d_{max}+1} - b^2}{b-1} \\
 &= \frac{1}{b-1} \frac{b^{d_{max}+1} - b^2}{b-1} = \frac{1}{b-1} N_b(d_{max})
 \end{aligned}$$

Tabelle 6.1 Vergleich der uninformatierten Suchalgorithmen. (*) bedeutet, dass die Aussage nur ist wahr bei konstanten Aktionskosten. ds ist die maximale Tiefe für einen endlichen Suchbaum

	Breite zuerst suchen	Einheitliche Kosten suchen	Tiefenorientiert suchen	Iterativ Vertiefung
Vollständigkeit	Ja	Ja	NEIN	Ja
Optimale Lösung	Ja (*)	Ja	NEIN	Ja (*)
Rechenzeit	d b	d b	ȳ oder b ds	B ^D
Speichernutzung	B ^D	B ^D	bd	bd

Knoten. Für $b > 2$ ist dies kleiner als die Anzahl $Nb(dmax)$ der Knoten im letzten Baum.

Für $b = 20$ enthalten die ersten $dmax$ ȳ 1 Bäume zusammen nur etwa $= 1/19$ der Anzahl der Knoten im letzten Baum. Die Rechenzeit für alle Iterationen außer

Letzte kann ignoriert werden.

Genau wie die Breitensuche ist diese Methode vollständig und mit konstanten Kosten verbunden Für alle Aktionen wird die kürzeste Lösung gefunden.

6.2.4 Vergleich

Die beschriebenen Suchalgorithmen sind in Tabelle 6.1 einander gegenübergestellt .

Wir können deutlich erkennen, dass die iterative Vertiefung der Gewinner dieses Tests ist, weil sie erhält in allen Kategorien die beste Note. Tatsächlich ist es von allen vier vorgestellten Algorithmen der nur praktisch verwendbar.

Wir haben tatsächlich einen Testsieger, wenn auch für realistische Anwendungen ist in der Regel nicht erfolgreich. Sogar für das 15er-Puzzle, den großen Bruder des 8er-Puzzles (siehe Aufgabe 6.4 auf Seite 110) gibt es etwa 2×10^{13} verschiedene Zustände. Für nicht trivial Bei Inferenzsystemen ist der Zustandsraum um viele Größenordnungen größer. Wie gezeigt in Sekte. 6.1, die ganze Rechenleistung der Welt wird nicht viel mehr helfen. Stattdessen Was benötigt wird, ist eine intelligente Suche, die nur einen winzigen Teil der Suche untersucht Raum und findet dort eine Lösung.

6.3 Heuristische Suche

Heuristiken sind Problemlösungsstrategien, die in vielen Fällen schneller zu einer Lösung führen als uninformierte Suche. Dies ist jedoch nicht garantiert. Heuristische Suche könnte erfordern viel mehr Zeit und können sogar dazu führen, dass die Lösung nicht gefunden wird.

Wir Menschen nutzen heuristische Verfahren erfolgreich für alles Mögliche. Wann Wenn wir zum Beispiel Gemüse im Supermarkt kaufen, beurteilen wir die verschiedenen Möglichkeiten für ein Pfund Erdbeeren anhand einiger einfacher Kriterien wie Preis, Aussehen, Produktionsquelle und Vertrauen in den Verkäufer, und dann entscheiden wir uns für die beste Option nach Bauchgefühl. Theoretisch könnte es besser sein, die Erdbeeren einer Grundierung zu unterziehen chemische Analyse, bevor Sie sich für den Kauf entscheiden. Beispielsweise könnten die Erdbeeren vergiftet sein. Wenn das der Fall wäre, hätte sich die Analyse gelohnt

```

HEURISTIKSEARCH(Start, Ziel)
NodeList = [Start]
Während wahr
    Wenn NodeList = ∅ Return("Keine Lösung")
    Knoten = First(NodeList)
    NodeList = Rest(NodeList)
    If GoalReached(Node, Goal) Return(„Lösung gefunden“, Node)
    NodeList = SortIn(Successors(Node),NodeList)

```

Abb. 6.12 Der Algorithmus für die heuristische Suche

Problem. Eine solche Analyse führen wir jedoch nicht durch, da die Wahrscheinlichkeit sehr hoch ist, dass unsere heuristische Auswahl gelingt und wir schnell an unser Ziel, leckere Erdbeeren zu essen, gelangen.

Heuristische Entscheidungen sind eng mit der Notwendigkeit verbunden, *Entscheidungen in Echtzeit mit begrenzten Ressourcen zu treffen*. In der Praxis wird eine schnell gefundene gute Lösung einer Lösung vorgezogen, die optimal ist, deren Ableitung aber sehr teuer ist.

Zur mathematischen Modellierung einer Heuristik wird eine heuristische Bewertungsfunktion $f(s)$ für Zustände verwendet. Ziel ist es, mit geringem Aufwand eine Lösung für das gestellte Suchproblem mit minimalen Gesamtkosten zu finden. Bitte beachten Sie, dass es einen subtilen Unterschied zwischen dem Aufwand, eine Lösung zu finden, und den Gesamtkosten dieser Lösung gibt. Beispielsweise kann es eine halbe Sekunde dauern, bis Google Maps eine Route vom Rathaus in San Francisco nach Tuolumne Meadows im Yosemite-Nationalpark findet, aber die Fahrt von San Francisco nach Tuolumne Meadows mit dem Auto kann vier Stunden und etwas Geld dauern für Benzin usw. (Gesamtkosten).

Als nächstes modifizieren wir den Breitensuchalgorithmus, indem wir ihm die Bewertungsfunktion hinzufügen. Die aktuell geöffneten Knoten werden nicht mehr zeilenweise von links nach rechts erweitert, sondern entsprechend ihrer heuristischen Bewertung. Aus der Menge der offenen Knoten wird immer zuerst der Knoten mit der minimalen Bewertung erweitert. Dies wird dadurch erreicht, dass Knoten beim Erweitern sofort ausgewertet und in die Liste der offenen Knoten eintragen. Die Liste kann dann Knoten aus unterschiedlichen Tiefen im Baum enthalten.

Da die heuristische Auswertung von Zuständen für die Suche sehr wichtig ist, unterscheiden wir im Folgenden zwischen Zuständen und den ihnen zugeordneten Knoten. Der Knoten `con` speichert den Zustand und weitere für die Suche relevante Informationen, wie z. B. seine Tiefe im Suchbaum und die heuristische Bewertung des Zustands. Dies hat zur Folge, dass die Funktion „Nachfolger“, die die Nachfolger (Kinder) eines Knotens generiert, für diese Nachfolgerknoten sofort auch deren heuristische Bewertungen als Bestandteil jedes Knotens berechnen muss. Wir definieren den allgemeinen Suchalgorithmus HEURISTICSEARCH in Abb. 6.12.

Die Knotenliste wird mit den Startknoten initialisiert. Anschließend wird in der Schleife der erste Knoten aus der Liste entfernt und daraufhin geprüft, ob es sich um einen Lösungsknoten handelt. Wenn nicht, wird es um die Funktion „Nachfolger“ erweitert und seine Nachfolger zur Liste hinzugefügt

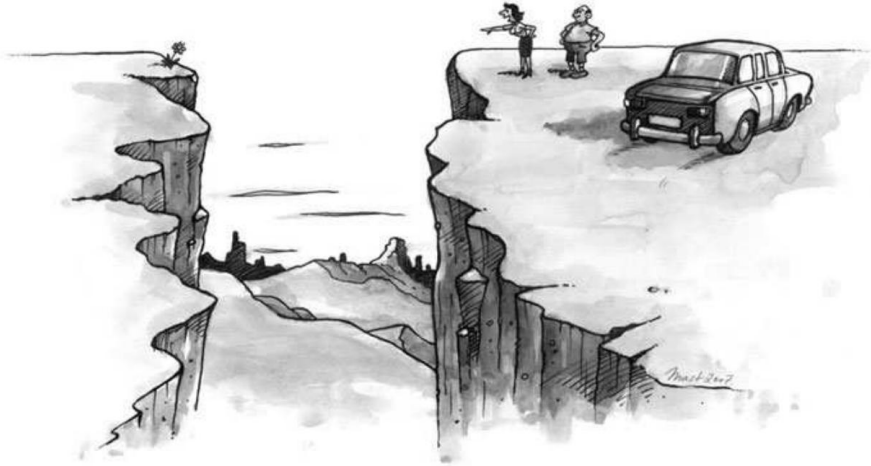


Abb. 6.13 Er: „Lieber, denken Sie an die Treibstoffkosten! Ich werde dir woanders eins pflücken.“ Sie: „Nein, das da drüben will ich haben!“

mit der Funktion „SortIn“. „SortIn(X,Y)“ fügt die Elemente aus der unsortierten Liste X in die aufsteigend sortierte Liste Y ein. Als Sortierschlüssel wird die heuristische Bewertung verwendet. Somit ist gewährleistet, dass der beste Knoten (also der mit dem niedrigsten heuristischen Wert) immer am Anfang der Liste steht.³ Auch

die Tiefensuche und die Breitensuche sind Sonderfälle der Funktion HEURISTICSEARCH. Wir können sie einfach generieren, indem wir die entsprechende Auswertungsfunktion einbinden (Übung 6.11 auf Seite 111).

Die beste Heuristik wäre eine Funktion, die die tatsächlichen Kosten von jedem Knoten bis zum Ziel berechnet. Dazu wäre allerdings eine Durchquerung des gesamten Suchraums erforderlich, was die Heuristik genau verhindern soll. Daher benötigen wir eine Heuristik, die schnell und einfach zu berechnen ist. Wie finden wir eine solche Heuristik?

Eine interessante Idee zum Finden einer Heuristik ist die Vereinfachung des Problems. Die ursprüngliche Aufgabe ist so vereinfacht, dass sie mit geringem Rechenaufwand gelöst werden kann. Die Kosten von einem Zustand zum Ziel im vereinfachten Problem dienen dann als Schätzung für das eigentliche Problem (siehe Abb. 6.13). Diese *Kostenschätzfunktion* bezeichnen wir mit h .

6.3.1 Gierige Suche

Es erscheint sinnvoll, aus der Liste der aktuell verfügbaren Zustände den Zustand mit dem niedrigsten geschätzten h -Wert (also den mit den niedrigsten geschätzten Kosten) auszuwählen. D

³Beim Einsortieren eines neuen Knotens aus der Knotenliste kann es von Vorteil sein, zu prüfen, ob der Knoten bereits verfügbar ist, und gegebenenfalls das Duplikat zu löschen.

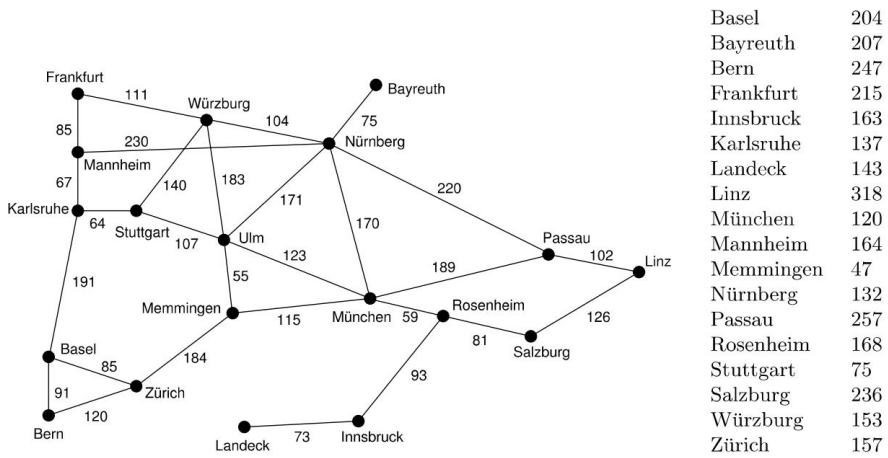


Abb. 6.14 Stadtdiagramm mit Flugentfernungen von allen Städten nach Ulm

Die Kostenschätzung kann dann direkt als Bewertungsfunktion verwendet werden. Für die Auswertung in der Funktion HEURISTICSEARCH setzen wir $f(s) = h(s)$. Dies ist am Beispiel der Reiseplanung (Beispiel 6.2 auf Seite 88) deutlich zu erkennen. Als Vereinfachung des Problems haben wir uns die Aufgabe gestellt, den geradlinigen Weg von Stadt zu Stadt (also die Flugstrecke) zu finden. Anstatt die optimale Route zu suchen, ermitteln wir zunächst von jedem Knoten aus eine Route mit minimaler Flugentfernung zum Ziel. Als Ziel wählen wir Ulm. Somit wird die Kostenschätzungsfunktion

$$h(s) = \text{Flugentfernung von Stadt } s \text{ nach Ulm.}$$

Die Flugentfernungen von allen Städten nach Ulm sind in Abb. 6.14 neben der Grafik dargestellt.

Der Suchbaum für den Start in Linz ist in Abb. 6.15 auf Seite 98 links dargestellt. Wir können sehen, dass der Baum sehr schlank ist. Die Suche ist somit schnell beendet. Leider findet diese Suche nicht immer die optimale Lösung. Beispielsweise gelingt es diesem Algorithmus beim Start in Mannheim nicht, die optimale Lösung zu finden (Abb. 6.15 auf Seite 98 rechts). Der Weg Mannheim–Nürnberg–Ulm hat eine Länge von 401 km. Die Strecke Mannheim–Karlsruhe–Stuttgart–Ulm wäre mit 238 km deutlich kürzer. Wenn wir die Grafik betrachten, wird die Ursache dieses Problems klar. Nürnberg liegt zwar etwas näher an Ulm als Karlsruhe, allerdings ist die Entfernung von Mannheim nach Nürnberg deutlich größer als die von Mannheim nach Karlsruhe. Die Heuristik blickt nur „gierig“ auf das Ziel voraus, anstatt auch die bereits zurückgelegte Strecke bis zum aktuellen Knoten zu berücksichtigen. Deshalb geben wir ihr den Namen *Greedy Search*.

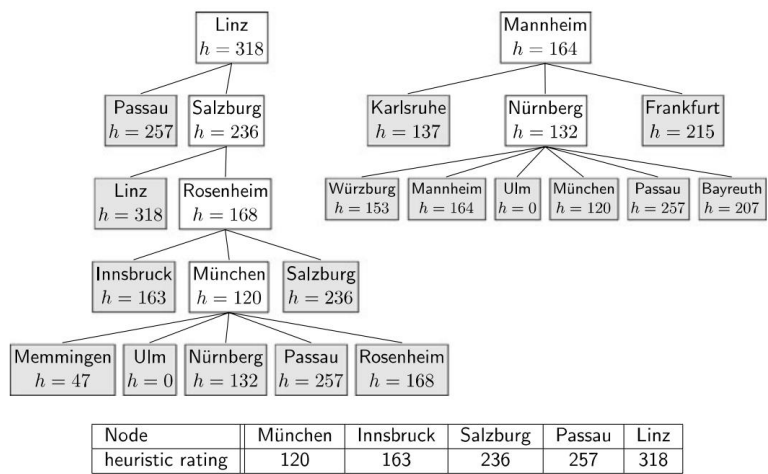


Abb. 6.15 Greedy-Suche: von Linz nach Ulm (*links*) und von Mannheim nach Ulm (*rechts*). Die Knotenlisten-Datenstruktur für den linken Suchbaum, sortiert nach der Knotenbewertung, bevor die Erweiterung des Knotens München angegeben wird

6.3.2 A* -Suche

Wir wollen nun die Kosten berücksichtigen, die bei der Suche bis zum aktuellen Knoten s entstanden sind. Zuerst definieren wir die *Kostenfunktion*

$g(s)$ = Summe der angefallenen Kosten vom Start bis zum aktuellen Knoten,

Fügen Sie dann die geschätzten Kosten für das Ziel hinzu und erhalten Sie die *heuristische Bewertungsfunktion*

$$f(s) = g(s) + h(s).$$

Nun fügen wir noch eine weitere kleine, aber wichtige Anforderung hinzu.

Definition 6.4 Eine heuristische Kostenschätzfunktion $h(s)$, die die tatsächlichen Kosten vom Zustand s zum Ziel niemals überschätzt, wird als *zulässig bezeichnet*.

Die Funktion HEURISTICSEARCH wird zusammen mit einer Bewertungsfunktion $f(s) = g(s) + h(s)$ und einer zulässigen heuristischen Funktion h *A* -Algorithmus genannt*. Dieser berühmte Algorithmus ist vollständig und optimal. A* findet somit immer die kürzeste Lösung für jedes lösbare Suchproblem. Wir werden dies in der folgenden Diskussion erklären und beweisen.

Zuerst wenden wir den A* -Algorithmus auf das Beispiel an. Wir suchen den Kürzesten Weg von Frankfurt nach Ulm.

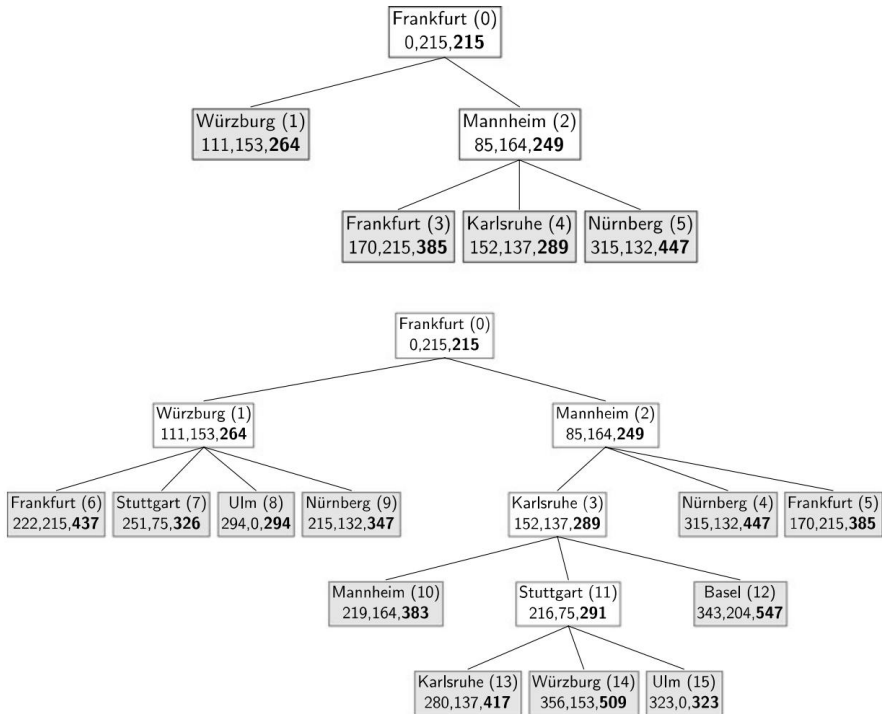


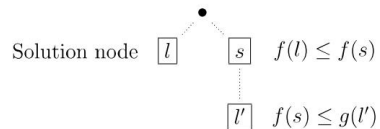
Abb. 6.16 Zwei Schnappschüsse des A^* -Suchbaums für die optimale Route von Frankfurt nach Ulm. In den Feldern unter dem Namen der Stadt s zeigen wir $g(s)$, $h(s)$, $f(s)$. Zahlen in Klammern hinter den Städtenamen zeigen die Reihenfolge, in der die Knoten durch die Funktion „Nachfolger“ generiert wurden.

Im oberen Teil von Abb. 6.16 sehen wir, dass die Nachfolger von Mannheim vor den Nachfolgern von Würzburg erzeugt werden. Die optimale Lösung Frankfurt–Würzburg–Ulm wird kurz darauf im achten Schritt generiert, aber noch nicht als solche erkannt. Somit terminiert der Algorithmus noch nicht, da der Knoten Karlsruhe (3) einen besseren (niedrigeren) f -Wert hat und somit in der Reihe vor dem Knoten Ulm (8) liegt. Erst wenn alle f -Werte größer oder gleich dem Lösungsknoten Ulm (8) sind, haben wir sichergestellt, dass wir eine optimale Lösung haben. Andernfalls könnte es möglicherweise eine andere Lösung mit geringeren Kosten geben. Wir werden nun zeigen, dass dies allgemein gilt.

Satz 6.2 Der A^* -Algorithmus ist optimal. Das heißt, es wird immer die Lösung mit den geringsten Gesamtkosten gefunden, wenn die Heuristik h zulässig ist.

Beweis Im HEURISTICSEARCH-Algorithmus wird jeder neu generierte Knoten s durch die Funktion „SortIn“ entsprechend seiner heuristischen Bewertung $f(s)$ einsortiert. Der Knoten mit dem kleinsten Bewertungswert steht somit am Anfang der Liste. Wenn der Knoten l am

Abb. 6.17 Der erste von A* gefundene Lösungsknoten l hat niemals höhere Kosten als ein anderer beliebiger Knoten s .



Wenn am Anfang der Liste ein Lösungsknoten steht, hat kein anderer Knoten eine bessere heuristische Bewertung. Für alle anderen Knoten s gilt dann $f(l) \leq f(s)$. Da die Heuristik zulässig ist, kann auch nach Erweiterung aller anderen Knoten keine bessere Lösung gefunden werden (siehe Abb. 6.17). Formal geschrieben:

$$g(l) = g(l) + h(l) = f(l) \leq f(s) = g(s) + h(s) \leq g(l').$$

Die erste Gleichheit gilt, weil l ein Lösungsknoten mit $h(l) = 0$ ist. Die zweite ist die Definition von f . Die dritte (Un-)Gleichheit gilt, weil die Liste der offenen Knoten in aufsteigender Reihenfolge sortiert ist. Die vierte Gleichung ist wiederum die Definition von f . Schließlich ist die letzte (Un-)Gleichheit die Zulässigkeit der Heuristik, die niemals die Kosten von Knoten zu einer beliebigen Lösung überschätzt. Somit wurde gezeigt, dass $g(l) \leq g(l')$, das heißt, dass die gefundene Lösung l optimal ist.

6.3.3 IDA* -Suche

Die A* -Suche übernimmt eine Eigenart der Breitensuche. Es müssen viele Knoten im Speicher gespeichert werden, was zu einer sehr hohen Speichernutzung führen kann. Darüber hinaus muss die Liste der offenen Knoten sortiert werden. Das Einfügen von Knoten in die Liste und das Entfernen von Knoten aus der Liste kann daher nicht mehr in konstanter Zeit erfolgen, was die Komplexität des Algorithmus leicht erhöht. Basierend auf dem Heapsort-Algorithmus können wir die Knotenliste als Heap mit logarithmischer Zeitkomplexität für das Einfügen und Entfernen von Knoten strukturieren (siehe [CLR90]).

Beide Probleme können – ähnlich wie bei der Breitensuche – durch iterative Vertiefung gelöst werden. Wir arbeiten mit der Tiefensuche und heben das Limit sukzessive an. Anstatt jedoch mit einem Tiefenlimit zu arbeiten, verwenden wir hier ein Limit für die heuristische Auswertung $f(s)$. Dieser Vorgang wird als IDA* -Algorithmus bezeichnet.

6.3.4 Empirischer Vergleich der Suchalgorithmen

In A* , oder (alternativ) IDA* , Wir haben einen Suchalgorithmus mit vielen guten Eigenschaften. Es ist vollständig und optimal. Es kann somit ohne Risiko verwendet werden. Das Wichtigste ist jedoch, dass es mit Heuristiken arbeitet und so die Rechenzeit zur Lösungsfindung deutlich reduzieren kann. Wir möchten dies am Beispiel des 8-Puzzles empirisch untersuchen.

Für das 8-Puzzle gibt es zwei einfache zulässige Heuristiken. Die Heuristik h_1 zählt einfach die Anzahl der Quadrate, die nicht an der richtigen Stelle sind. Diese Heuristik ist eindeutig zulässig. Die Heuristik h_2 misst die *Manhattan-Entfernung*. Für jeden

Tabelle 6.2 Vergleich des Rechenaufwands der uninformierten Suche und der heuristischen Suche nach lösbbare 8-Puzzle-Aufgaben mit unterschiedlicher Tiefe. Die Messungen erfolgen in Schritten und Sekunden. Alle Werte sind Durchschnittswerte über mehrere Läufe (siehe letzte Spalte)

Tiefe	Iterative		Äy- Algorithmus				Num. läuft
	Vertiefungsschritte	Zeit [Sek.]	Heuristik h1		Heuristik h2		
			Schritte	Zeit [Sek.]	Schritte	Zeit [Sek.]	
2	20	0,003	3,0	0,0010	3,0	0,0010	10
4	81	0,013	5.2	0,0015	5,0	0,0022	24
6	806	0,13	10.2	0,0034	8.3	0,0039	19
8	6455	1,0	17.3	0,0060	12.2	0,0063	14
10	50512	7.9	48.1	0,018	22.1	0,011	15
12	486751	75,7	162.2	0,074	56,0	0,031	12
IDAy							
14	–	–	10079.2	2.6	855,6	0,25	16
16	–	–	69386.6	19.0	3806.5	1.3	13
18	–	–	708780,0	161,6	53941,5	14.1	4

Quadrieren Sie die horizontalen und vertikalen Abstände zur Position dieses Quadrats im Zielzustand werden addiert. Dieser Wert wird dann über alle Quadrate summiert. Zum Beispiel die Manhattan-Entfernung der beiden Staaten

2	5	
1	4	8
7	3	6

and

1	2	3
4	5	6
7	8	

wird berechnet als

$h2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$

Auch die Zulässigkeit der Manhattan-Distanz liegt auf der Hand (siehe Aufgabe 6.13). Seite 111).

Die beschriebenen Algorithmen wurden in Mathematica implementiert. Zum Vergleich bei uninformierter Suche der Äy- Algorithmus mit den beiden Heuristiken h1 und h2 und Die iterative Vertiefung wurde auf 132 zufällig generierte 8-Puzzle-Aufgaben angewendet. Der Durchschnittswerte für die Anzahl der Schritte und die Rechenzeit sind in Tabelle 6.2 angegeben. Wir sehen, dass die Heuristiken die Suchkosten im Vergleich zur uninformierten Suche erheblich reduzieren.

Wenn wir beispielsweise die iterative Vertiefung mit Äy mit h1 in Tiefe 12 vergleichen , ist es Es wird deutlich, dass h1 die Anzahl der Schritte um etwa den Faktor 3.000 reduziert, aber die Rechenzeit nur um den Faktor 1.023. Dies liegt an den höheren Kosten pro Schritt zur Berechnung der Heuristik.

Bei näherer Betrachtung erkennt man einen Sprung in der Anzahl der Schritte zwischen Tiefe 12 und Tiefe 14 in der Spalte für h1. Dieser Sprung lässt sich nicht allein durch das Wiederholen erklären Arbeit von IDAy · Es entsteht durch die Implementierung des Äy- Algorithmus

löscht Duplikate identischer Knoten und verkleinert dadurch den Suchraum. Dies ist mit IDA* nicht möglich, da es fast keine Knoten speichert. Trotzdem kann A* über die Tiefe 14 hinaus nicht mehr mit IDA* konkurrieren, da die Kosten für das Sortieren in neuen Knoten die Zeit pro Schritt so stark in die Höhe treiben.

Eine Berechnung des effektiven Verzweigungsfaktors nach (6.1) auf Seite 87 ergibt Werte von etwa 2,8 für die uninformierte Suche. Diese Zahl deckt sich mit dem Wert aus Abschn. 6.1. Die Heuristik h_1 reduziert den Verzweigungsfaktor auf Werte von etwa 1,5 und h_2 auf etwa 1,3. Wir können in der Tabelle sehen, dass eine kleine Reduzierung des Verzweigungsfaktors von 1,5 auf 1,3 uns einen großen Vorteil bei der Rechenzeit verschafft.

Die heuristische Suche hat daher eine wichtige praktische Bedeutung, da sie Probleme lösen kann, die für eine uninformierte Suche weit außerhalb der Reichweite liegen.

6.3.5 Zusammenfassung

Von den verschiedenen Suchalgorithmen für die uninformierte Suche ist die iterative Vertiefung der einzig praktikable, da sie vollständig ist und mit sehr wenig Speicher auskommt.

Bei schwierigen kombinatorischen Suchproblemen scheitert jedoch selbst eine iterative Vertiefung meist an der Größe des Suchraums. Hier hilft die heuristische Suche durch die Reduzierung des effektiven Verzweigungsfaktors. Der IDA*-Algorithmus ist ebenso wie die iterative Vertiefung vollständig und benötigt sehr wenig Speicher.

Heuristiken bringen natürlich nur dann einen signifikanten Vorteil, wenn die Heuristik „gut“ ist. Bei der Lösung schwieriger Suchprobleme besteht die eigentliche Aufgabe des Entwicklers darin, Heuristiken zu entwerfen, die den effektiven Verzweigungsfaktor stark reduzieren. In Abschn. In Abschn. 6.5 gehen wir auf dieses Problem ein und zeigen außerdem, wie Techniken des maschinellen Lernens zur automatischen Generierung von Heuristiken eingesetzt werden können.

Abschließend bleibt festzuhalten, dass Heuristiken bei unlösbaren Problemen keinen Leistungsvorteil haben, da die Unlösbarkeit eines Problems erst festgestellt werden kann, wenn der gesamte Suchbaum durchsucht wurde. Für entscheidbare Probleme wie das 8-Puzzle bedeutet dies, dass der gesamte Suchbaum bis zu einer maximalen Tiefe durchlaufen werden muss, unabhängig davon, ob eine Heuristik verwendet wird oder nicht. Die Heuristik ist in diesem Fall immer ein Nachteil, was auf den Rechenaufwand für die Auswertung der Heuristik zurückzuführen ist. Dieser Nachteil kann in der Regel durch einen konstanten Faktor abgeschätzt werden, der von der Größe des Problems unabhängig ist. Für unentscheidbare Probleme wie den Beweis von PL1-Formeln kann der Suchbaum unendlich tief sein. Das bedeutet, dass die Suche im unlösbaren Fall möglicherweise nie endet. Zusammenfassend lässt sich Folgendes sagen: Bei lösbaren Problemen reduzieren Heuristiken die Rechenzeit oft drastisch, bei unlösbaren Problemen kann der Aufwand bei Heuristiken jedoch sogar höher sein.

6.4 Spiele mit Gegnern

Spiele für zwei Spieler, wie Schach, Dame, Othello und Go, sind deterministisch, da jede Aktion (ein Zug) bei gleichem übergeordneten Zustand zum gleichen untergeordneten Zustand führt. Im Gegensatz dazu ist Backgammon nicht deterministisch, da sein untergeordneter Zustand davon abhängt

auf das Ergebnis eines Würfelwurfs. Diese Spiele sind alle beobachtbar, da jeder Spieler immer den kompletten Spielstand kennt. Viele Kartenspiele, wie zum Beispiel Poker, sind nur teilweise beobachtbar, weil der Spieler die Karten der anderen Spieler nicht oder nur teilweise kennt.

Die bisher in diesem Kapitel diskutierten Probleme waren deterministisch und beobachtbar. Im Folgenden betrachten wir Spiele, die ebenfalls deterministisch und beobachtbar sind. Darüber hinaus beschränken wir uns auf Nullsummenspiele. Dabei handelt es sich um Spiele, bei denen jeder Gewinn eines Spielers einen gleichwertigen Verlust für den Gegner bedeutet. Die Summe aus Gewinn und Verlust ist immer gleich Null. Dies gilt für die oben erwähnten Spiele Schach, Dame, Othello und Go.

6.4.1 Minimax-Suche

Das Ziel jedes Spielers ist es, optimale Züge zu machen, die zum Sieg führen. Im Prinzip ist es möglich, einen Suchbaum zu konstruieren und diesen (wie beim 8er-Puzzle) vollständig nach einer Reihe von Zügen zu durchsuchen, die zum Sieg führen. Es gibt jedoch einige Besonderheiten, auf die man achten

sollte: 1. Der effektive Verzweigungsfaktor im Schach liegt bei etwa 30 bis 35. In einem typischen Spiel mit 50 Zügen pro Spieler hat der Suchbaum mehr als $30^{100} \approx 10^{148}$ Blattknoten. Daher besteht keine Möglichkeit, den Suchbaum vollständig zu erkunden. Außerdem wird Schach oft mit Zeitlimit gespielt. Aufgrund dieser *Echtzeitanforderung* muss die Suche auf eine geeignete Tiefe im Baum beschränkt werden, beispielsweise auf acht Halbzüge. Da es unter den Blattknoten dieses tiefenbegrenzten Baums normalerweise keine Lösungsknoten (also Knoten, die das Spiel beenden) gibt, wird eine heuristische *Bewertungsfunktion* B für Brettpositionen verwendet. Der Spielgrad des Programms hängt stark von der Qualität dieser Bewertungsfunktion ab. Daher werden wir dieses Thema in Abschn. 6.5.

2. Im Folgenden nennen wir den Spieler, dessen Spiel wir optimieren wollen, Max und seinen Gegner Min. Die Bewegungen des Gegners (Min) sind nicht im Voraus bekannt, und daher ist auch der eigentliche Suchbaum nicht bekannt. Dieses Problem lässt sich elegant lösen, indem man annimmt, dass der Gegner immer den bestmöglichen Zug macht. Je höher die Bewertung $B(s)$ für Position s ist, desto besser ist Position s für den Spieler Max und desto schlechter ist sie für seinen Gegner Min. Max versucht, die Bewertung seiner Züge zu maximieren, während Min Bewegungen ausführt, die zu einer möglichst niedrigen Bewertung führen.

Ein Suchbaum mit vier Halbbewegungen und Bewertungen aller Blätter ist in Abb. 6.18 auf Seite 104 dargestellt. Die Bewertung eines inneren Knotens wird rekursiv als Maximum oder Minimum seiner untergeordneten Knoten abgeleitet, abhängig von der Ebene des Knotens.

6.4.2 Alpha-Beta-Beschneidung

Durch den Wechsel zwischen Maximierung und Minimierung können wir uns unter Umständen viel Arbeit ersparen. Das Alpha-Beta-Pruning funktioniert mit der Tiefensuche bis zu einer voreingestellten Tiefengrenze. Auf diese Weise wird der Suchbaum von links durchsucht

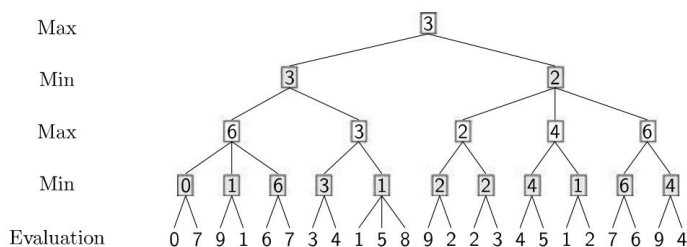


Abb. 6.18 Ein Minimax-Spielbaum mit Vorausschau auf vier Halbzüge

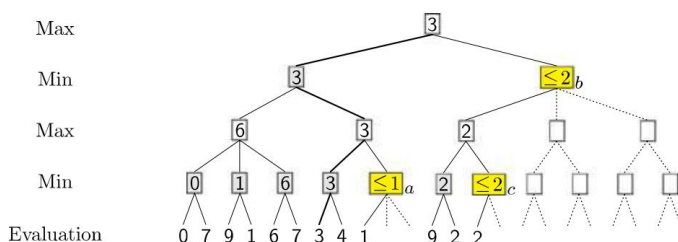


Abb. 6.19 Ein Alpha-Beta-Spielbaum mit Vorausschau auf vier Halbzüge. Die gepunkteten Teile des Baums werden nicht durchlaufen, da sie keinen Einfluss auf das Endergebnis haben

nach rechts. Wie bei der Minimax-Suche wird bei den Minimum-Knoten das Minimum aus dem Minimalwert der Nachfolgerknoten generiert, bei den Maximum-Knoten ebenfalls

das Maximum. In Abb. 6.19 ist dieser Vorgang für den Baum aus Abb. 6.18 dargestellt. An dem mit a gekennzeichneten Knoten können alle anderen Nachfolger ignoriert werden, nachdem das erste Kind als Wert 1 ausgewertet wurde, da das Minimum mit Sicherheit ≥ 1 ist. Es könnte sogar noch kleiner werden, aber das spielt keine Rolle, da das Maximum eine Ebene darüber bereits ≥ 3 liegt. Unabhängig davon, wie die Bewertung der verbleibenden Nachfolger ausfällt, behält das Maximum den Wert 3. Analog wird der Baum am Knoten b beschnitten. Da das erste Kind von b den Wert 2 hat, kann das zu erzeugende Minimum für b nur kleiner oder gleich 2 sein. Das Maximum am Wurzelknoten ist aber schon sicher ≥ 3 .

Dies kann durch Werte ≥ 2 nicht geändert werden. Somit können die verbleibenden Teilbäume von b beschnitten werden.

Die gleiche Argumentation gilt für den Knoten c. Allerdings ist der relevante maximale Knoten ist nicht der direkte Elternknoten, sondern der Wurzelknoten. Dies lässt sich

verallgemeinern. • An jedem Blattknoten wird die Bewertung

berechnet. • Für jeden Maximumknoten wird der aktuell größte Kindwert in $\hat{y}$ gespeichert.

• Für jeden minimalen Knoten wird der aktuell kleinste Kindwert in $\hat{y}$ gespeichert. • Wenn

an einem minimalen Knoten k der aktuelle Wert $\hat{y} \geq \hat{y}$ ist, kann die Suche unter k beendet werden.

Dabei ist $\hat{y}$ der größte Wert eines maximalen Knotens im Pfad von der Wurzel nach k. • Wenn

an einem maximalen Knoten l der aktuelle Wert $\hat{y} \leq \hat{y}$ ist, kann die Suche unter l beendet werden.

Dabei ist $\hat{y}$ der kleinste Wert eines minimalen Knotens im Pfad von der Wurzel nach l.

```

ALPHABETAMAX(Knoten,  $\tilde{y}$ ,  $\check{y}$ )

If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Nachfolger(Knoten)
Während NewNodes =  $\tilde{y}$ 
     $\check{y}$  = Maximum( $\check{y}$ , ALPHABETAMIN(First(NewNodes),  $\tilde{y}$ ,  $\check{y}$ ))
    Wenn  $\tilde{y}$   $\check{y}$   $\tilde{y}$  Return( $\check{y}$ )
    NewNodes = Rest(NewNodes)
Rückgabe( $\check{y}$ )

```

```

ALPHABETAMIN(Knoten,  $\tilde{y}$ ,  $\check{y}$ )

If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Nachfolger(Knoten)
Während NewNodes =  $\check{y}$ 
     $\tilde{y}$  = Minimum( $\tilde{y}$ , ALPHABETAMAX(First(NewNodes),  $\tilde{y}$ ,  $\check{y}$ ))
    Wenn  $\tilde{y}$   $\check{y}$   $\tilde{y}$  Return( $\check{y}$ )
    NewNodes = Rest(NewNodes)
Rückkehr( $\tilde{y}$ )

```

Abb. 6.20 Der Algorithmus zur Alpha-Beta-Suche mit den beiden Funktionen ALPHABETAMIN und ALPHABETAMAX

Der in Abb. 6.20 dargestellte Algorithmus ist eine Erweiterung der Tiefensuche mit zwei Funktionen, die abwechselnd aufgerufen werden. Es verwendet die oben definierten Werte für $\tilde{y}$ und $\check{y}$.

Komplexität: Die durch Alpha-Beta-Pruning eingesparte Rechenzeit hängt stark von der Reihenfolge ab, in der untergeordnete Knoten durchlaufen werden. Im schlimmsten Fall bringt der Alpha-Beta-Schnitt keinen Vorteil. Für einen konstanten Verzweigungsfaktor b ist die Anzahl n_d der in der Tiefe d auszuwertenden Blattknoten gleich

$$n_d = b^d.$$

Im besten Fall, wenn die Nachfolger maximaler Knoten absteigend und die Nachfolger minimaler Knoten aufsteigend sortiert sind, reduziert sich der effektive Verzweigungsfaktor auf $\sqrt[b]{b}$. Im Schach bedeutet dies eine erhebliche Reduzierung des effektiven Verzweigungsfaktors von 35 auf etwa 6. Dann nur

$$n_d = \sqrt[b]{b}^d = b^{d/2}$$

Blattknoten würden erstellt. Das bedeutet, dass die Tiefenbegrenzung und damit auch der Suchhorizont beim Alpha-Beta-Pruning verdoppelt werden. Dies trifft jedoch nur bei optimal sortierten Nachfolgern zu, da die Bewertungen der untergeordneten Knoten zum Zeitpunkt ihrer Erstellung unbekannt sind. Wenn die untergeordneten Knoten zufällig sortiert sind, wird der Verzweigungsfaktor auf $b^{3/4}$ und die Anzahl der Blattknoten auf b reduziert

$$nd = b^{\frac{3}{4}d}.$$

Mit der gleichen Rechenleistung kann ein Schachcomputer mit Alpha-Beta-Pruning beispielsweise acht statt sechs Halbzüge voraus berechnen, mit einem effektiven Verzweigungsfaktor von etwa 14. Eine ausführliche Analyse mit einer Ableitung dieser Parameter finden Sie in [Pea84].

Um die Suchtiefe wie oben erwähnt zu verdoppeln, müssten die untergeordneten Knoten optimal angeordnet sein, was in der Praxis nicht der Fall ist. Sonst wäre die Suche unnötig. Mit einem einfachen Trick können wir eine relativ gute Knotenreihenfolge erreichen. Wir verbinden Alpha-Beta-Pruning mit iterativer Vertiefung über die Tiefengrenze hinaus. Somit können wir bei jeder neuen Tiefengrenze auf die Bewertungen aller Knoten vorheriger Ebenen zugreifen und die Nachfolger bei jedem Zweig ordnen. Dadurch erreichen wir einen effektiven Verzweigungsfaktor von etwa 7 bis 8, was nicht weit vom theoretischen O

6.4.3 Nichtdeterministische Spiele

Die Minimax-Suche kann auf alle Spiele mit nicht deterministischen Aktionen, wie z. B. Backgammon, verallgemeinert werden. Jeder Spieler würfelt vor seinem Zug, der durch das Ergebnis des Würfelwurfs beeinflusst wird. Im Spielbaum gibt es nun in der Reihenfolge drei Arten von Levels

Max, Würfel, Min, Würfel, . . . ,

wobei jeder Würfelwurfknoten sechs Wege verzweigt. Da wir den Wert des Würfels nicht vorhersagen können, mitteln wir die Werte aller Würfe und führen die Suche wie beschrieben mit den Durchschnittswerten aus [RN10] durch.

6.5 Heuristische Auswertungsfunktionen

Wie finden wir eine gute heuristische Bewertungsfunktion für die Suchaufgabe? Hier gibt es grundsätzlich zwei Ansätze. Der klassische Weg nutzt das Wissen menschlicher Experten. Dem Wissensingenieur wird die meist schwierige Aufgabe übertragen, das implizite Wissen des Experten in Form eines Computerprogramms zu formalisieren. Wie dieser Vorgang vereinfacht werden kann, wollen wir nun am Beispiel eines Schachprogramms zeigen.

Im ersten Schritt werden Experten zu den wichtigsten Faktoren bei der Umzugsauswahl befragt. Anschließend wird versucht, diese Faktoren zu quantifizieren. Wir erhalten eine Liste relevanter Merkmale oder Attribute. Diese werden dann (im einfachsten Fall) zusammengefasst

eine lineare Bewertungsfunktion $B(s)$ für Positionen, die wie folgt aussehen könnte:

$$B(s) = a_1 \cdot \text{Material} + a_2 \cdot \text{Bauernstruktur} + a_3 \cdot \text{Königssicherheit} \\ + a_4 \cdot \text{knight\_in\_center} + a_5 \cdot \text{Bishop\_diagonal\_coverage} + \dots, \quad (6.3)$$

wobei „Material“ das mit Abstand wichtigste Merkmal ist und nach berechnet wird

$$\text{material} = \text{material}(\text{own\_team}) - \text{material}(\text{gegner})$$

mit

$$\begin{aligned} \text{material}(\text{team}) = & \text{num\_pawns}(\text{team}) \cdot 100 + \text{num\_knights}(\text{team}) \cdot 300 \\ & + \text{num\_bishops}(\text{team}) \cdot 300 + \text{num\_rooks}(\text{team}) \cdot 500 \\ & + \text{num\_queens}(\text{team}) \cdot 900 \end{aligned}$$

Fast alle Schachprogramme nehmen eine ähnliche Materialbewertung vor. Allerdings gibt es bei allen anderen Features große Unterschiede, auf die wir hier nicht näher eingehen [Fra05, Lar00].

Im nächsten Schritt müssen die Gewichte a_i aller Merkmale bestimmt werden. Diese werden nach Diskussion mit Experten intuitiv festgelegt und dann nach jedem Spiel aufgrund positiver und negativer Erfahrungen geändert. Die Tatsache, dass dieser Optimierungsprozess sehr aufwändig ist und darüber hinaus die lineare Kombination von Merkmalen sehr begrenzt ist, legt den Einsatz von maschinellem Lernen nahe.

6.5.1 Erlernen von Heuristiken

Wir wollen nun die Gewichte a_i der Bewertungsfunktion $B(s)$ aus (6.3) automatisch optimieren. Bei diesem Ansatz wird der Experte lediglich nach den relevanten Merkmalen $f_1(s)$, ... befragt. ... $f_n(s)$ für Spielstatus s . Anschließend kommt ein maschinelles Lernverfahren zum Einsatz mit dem Ziel, eine möglichst optimale Bewertungsfunktion zu finden. Wir beginnen mit einer zunächst voreingestellten Bewertungsfunktion (bestimmt durch den Lernprozess) und lassen dann das Schachprogramm spielen. Am Ende des Spiels wird aus dem Ergebnis (Sieg, Niederlage oder Unentschieden) eine Wertung abgeleitet. Basierend auf dieser Bewertung wird die Bewertungsfunktion geändert mit dem Ziel, beim nächsten Mal weniger Fehler zu machen. Im Prinzip wird nun das Gleiche, was der Entwickler tut, automatisch durch den Lernprozess erledigt.

So einfach das klingt, so schwierig ist es in der Praxis. Ein zentrales Problem bei der Verbesserung der Positionsbewertung anhand gewonnener oder verllorener Spiele ist heute als *Credit-Assignment*- Problem bekannt. Wir haben zwar eine Wertung am Ende der Partie, aber keine Wertung für die einzelnen Spielzüge. Der Agent führt also viele Aktionen aus, erhält aber bis zum Schluss weder positives noch negatives Feedback. Wie soll es dieses Feedback dann den vielen in der Vergangenheit ergriffenen Maßnahmen zuordnen? Und wie sollte es in diesem Fall sein Handeln verbessern? Mit solchen Fragen beschäftigt sich das spannende junge Feld des *Reinforcement Learning* (siehe Abschn. 10).

Die meisten der besten Schachcomputer der Welt funktionieren heute noch ohne Techniken des maschinellen Lernens. Dafür gibt es zwei Gründe. Einerseits benötigen die bisher entwickelten Reinforcement-Learning-Algorithmen bei großen Zustandsräumen sehr viel Rechenzeit. Andererseits sind die manuell erstellten Heuristiken von Hochleistungsschachcomputern bereits stark optimiert. Das bedeutet, dass nur ein sehr gutes Lernsystem zu Verbesserungen führen kann. In den nächsten zehn Jahren wird voraussichtlich die Zeit kommen, in der ein lernender Computer Weltmeister wird.

6.6 Stand der Technik

Zur Bewertung der Qualität der heuristischen Suchprozesse möchte ich noch einmal wiederholen Elaine Richs Definition [Ric83]:

Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, Dinge zu tun, in denen Menschen derzeit besser sind.

Es gibt kaum einen besser geeigneten Test zur Entscheidung, ob ein Computerprogramm intelligent ist, als den direkten Vergleich von Computer und Mensch in einem Spiel wie Schach, Dame, Backgammon oder Go.

1950 stellten Claude Shannon, Konrad Zuse und John von Neumann die ersten Schachprogramme vor, deren Umsetzung jedoch entweder nicht oder nur mit sehr viel Zeitaufwand umsetzbar war. Nur wenige Jahre später, im Jahr 1955, schrieb Arthur Samuel ein Programm, das Dame spielte und durch einen einfachen Lernprozess seine eigenen Parameter verbessern konnte. Dazu nutzte er den ersten programmierbaren Logikrechner, den IBM 701. Im Vergleich zu heutigen Schachcomputern verfügte er jedoch über Zugriff auf eine Vielzahl archivierter Partien, bei denen jeder einzelne Zug von Experten bewertet worden war. Dadurch verbesserte das Programm seine Auswertungsfunktion. Um weitere Verbesserungen zu erzielen, ließ Samuel sein Programm gegen sich selbst spielen. Er hat das Problem der Kreditvergabe auf einfache Weise gelöst. Für jede einzelne Position während eines Spiels vergleicht es die Bewertung durch die Funktion $B(s)$ mit der durch Alpha-Beta-Pruning berechneten und ändert $B(s)$ entsprechend. 1961 schlug sein Dameprogramm den viertbesten Damespieler der USA. Mit dieser bahnbrechenden Arbeit war Samuel seiner Zeit sicherlich fast 30 Jahre voraus.

Erst Anfang der neunziger Jahre, als Reinforcement Learning aufkam, baute Gerald Tesauro ein Lern-Backgammonprogramm namens TD-Gammon auf, das auf Weltmeisterniveau spielte (siehe Abschn. 10).

Heutzutage gibt es mehrere Schachprogramme, die auf Großmeisterniveau spielen, und einige werden kommerziell für den PC verkauft. Der Durchbruch gelang 1997, als IBMs Deep Blue den Schachweltmeister Gary Kasparov mit einer Punktzahl von 3,5 zu 2,5 besiegte. Deep Blue konnte mit Alpha-Beta-Bereinigung und heuristischer Stellungsauswertung durchschnittlich 12 Halbzüge voraus berechnen.

Einer der bislang leistungsstärksten Schachcomputer ist Hydra, ein Parallelcomputer einer Firma in den Vereinigten Arabischen Emiraten. Die Software wurde von den Wissenschaftlern Christian Donninger (Österreich) und Ulf Lorenz (Deutschland) sowie dem deutschen Schachgroßmeister Christopher Lutz entwickelt. Hydra verwendet 64 parallele Xeon

Prozessoren mit ca. 3 GHz Rechenleistung und jeweils 1 GByte Speicher. Für die Positionsauswertung verfügt jeder Prozessor über einen FPGA-Coprozessor (Field Programmable Gate Array). Dadurch wird es möglich, auch mit einer teuren Auswertefunktion 200 Millionen Positionen pro Sekunde auszuwerten.

Mit dieser Technologie kann Hydra im Durchschnitt etwa 18 Züge im Voraus berechnen. In besonderen, kritischen Situationen kann der Suchhorizont sogar auf 40 Halbzüge ausgedehnt werden. Offensichtlich übersteigt diese Art von Horizont selbst große Champions, denn Hydra macht oft Bewegungen, die große Champions nicht verstehen können, die aber am Ende zum Sieg führen. Im Jahr 2005 besiegte Hydra den siebtplatzierten Großmeister Michael Adams mit 5,5 zu 0,5 Spielen.

Hydra verwendet wenig spezielles Lehrbuchwissen über Schach, eher eine Alpha-Beta-Suche mit relativ allgemeinen, bekannten Heuristiken und einer guten handcodierten Stellungsbewertung. Insbesondere ist Hydra nicht lernfähig. Zwischen den Spielen werden von den Entwicklern Verbesserungen vorgenommen. Somit wird der Computer weiterhin vom Lernen entlastet. Hydra verfügt auch über keine speziellen Planungsalgorithmen. Die Tatsache, dass Hydra ohne Lernen funktioniert, ist ein Hinweis darauf, dass trotz vieler Fortschritte noch Forschungsbedarf im Bereich maschinelles Lernen besteht. Darauf gehen wir, wie oben erwähnt, im Abschn. 10 und Kap. 8.

Im Jahr 2009 gewann das auf einem PDA laufende System Pocket Fritz 4 das Copa Mercosur-Schachturnier in Buenos Aires mit neun Siegen und einem Unentschieden gegen zehn hervorragende menschliche Schachspieler, darunter drei Großmeister. Auch wenn nicht viele Informationen über den internen Aufbau der Software verfügbar sind, stellt diese Schachmaschine einen Trend weg von reiner Rechenleistung hin zu mehr Intelligenz dar. Diese Maschine spielt auf Großmeisterniveau und ist mit Hydra vergleichbar, wenn nicht besser. Laut Pocket Fritz-Entwickler Stanislav Tsukrov [Wik10] durchsucht Pocket Fritz mit seiner Schachsuchmaschine HIARCS 13 weniger als 20.000 Positionen pro Sekunde und ist damit etwa um den Faktor 10.000 langsamer als Hydra. Dies führt zu dem Schluss, dass HIARCS 13 definitiv bessere Heuristiken zur Verringerung des effektiven Verzweigungsfaktors verwendet als Hydra und daher durchaus als intelligenter als Hydra bezeichnet werden kann. HIARCS ist übrigens eine Abkürzung für *Higher Intelligence Auto Response C*.

Auch wenn bald kein Mensch mehr eine Chance gegen die besten Schachcomputer haben wird, gibt es noch viele Herausforderungen für die KI. Gehen Sie zum Beispiel. Bei diesem alten japanischen Spiel, das auf einem quadratischen Brett mit 361 Feldern, 181 weißen und 180 schwarzen Steinen gespielt wird, beträgt der effektive Verzweigungsfaktor etwa 300. Nach vier Halbzügen sind es bereits etwa 8×10^9 Stellungen. Alle bekannten klassischen Spielbaum-Suchverfahren haben auf diesem Komplexitätsniveau keine Chance gegen gute menschliche Go-Spieler. Die Experten sind sich einig, dass es hier „wirklich intelligente“ Systeme braucht. Die kombinatorische Aufzählung aller Möglichkeiten ist die falsche Methode. Vielmehr werden Prozesse benötigt, die Muster auf dem Board erkennen, langfristige Entwicklungen verfolgen und schnell „intuitive“ Entscheidungen treffen können. Ähnlich wie bei der Erkennung von Objekten in komplexen Bildern ist der Mensch Computerprogrammen immer noch meilenweit voraus. Wir verarbeiten das Bild als Ganzes hochgradig parallel, während der Computer die Millionen Pixel nacheinander verarbeitet und es schwer hat, in der Fülle der Pixel das Wesentliche zu erkennen. Das Programm „The Many Faces of Go“ kann 1.100 verschiedene Muster erkennen und kennt 200 Spielstrategien. Aber alle Go-Programme haben immer noch

6.7 Übungen

Aufgabe 6.1

- (a) Beweisen Sie Satz 6.1 auf Seite 88, mit anderen Worten, beweisen Sie, dass sich für einen Baum mit großem konstanten Verzweigungsfaktor b fast alle Knoten auf der letzten Ebene in der Tiefe d befinden. (b) Zeigen Sie, dass dies nicht immer zutrifft, wenn der effektive Verzweigungsfaktor groß und nicht konstant ist.

Aufgabe 6.2

- (a) Berechnen Sie den durchschnittlichen Verzweigungsfaktor für das 8-Puzzle ohne Prüfung auf Zyklen. Der durchschnittliche Verzweigungsfaktor ist der Verzweigungsfaktor, den ein Baum mit einer gleichen Anzahl von Knoten auf der letzten Ebene, einem konstanten Verzweigungsfaktor und gleicher Tiefe hätte. (b) Berechnen Sie den durchschnittlichen Verzweigungsfaktor für das 8-Rätsel für die uninformierte Suche unter Vermeidung von Zyklen der Länge 2.

Aufgabe 6.3

- (a) Was ist der Unterschied zwischen dem Durchschnitt und dem effektiven Verzweigungsfaktor? (Definition 6.2 auf Seite 87)? (b) Warum eignet sich der effektive Verzweigungsfaktor besser zur Analyse und zum Vergleich der Rechenzeit von Suchalgorithmen als der durchschnittliche Verzweigungsfaktor? (c) Zeigen Sie, dass für einen stark verzweigten Baum mit n Knoten und der Tiefe d der effektive Verzweigungsfaktor b^* ungefähr gleich dem durchschnittlichen Verzweigungsfaktor und damit gleich $\sqrt[d]{n}$ ist.

Aufgabe 6.4

- (a) Berechnen Sie die Größe des Zustandsraums für das 8-Puzzle, für das analoge 3-Puzzle (2×2 -Matrix) sowie für das 15-Puzzle (4×4 -Matrix). (b) Beweisen Sie, dass der Zustandsgraph, der aus den Zuständen (Knoten) und den Aktionen (Kanten) für das 3-Puzzle besteht, in zwei verbundene Teilgraphen zerfällt, zwischen denen es keine Verbindungen gibt.

Übung 6.5 Finden Sie mit der Breitensuche für das 8-Rätsel einen Pfad (manuell) von

der Startknoten

1		3
4	2	6
7	5	8

 zum Zielknoten

1	2	3
4	5	6
7	8	

.

Übung 6.6

- (a) Programmieren Sie die Breitensuche, die Tiefensuche und die iterative Vertiefung in der Sprache Ihrer Wahl und testen Sie sie am Beispiel mit 8 Rätseln. (b) Warum macht es wenig Sinn, beim 8-Puzzle die Tiefensuche zu verwenden?

Aufgabe 6.7

- (a) Zeigen Sie, dass die Breitensuche bei konstanten Kosten für alle Aktionen garantiert ist um die kürzeste Lösung zu finden.

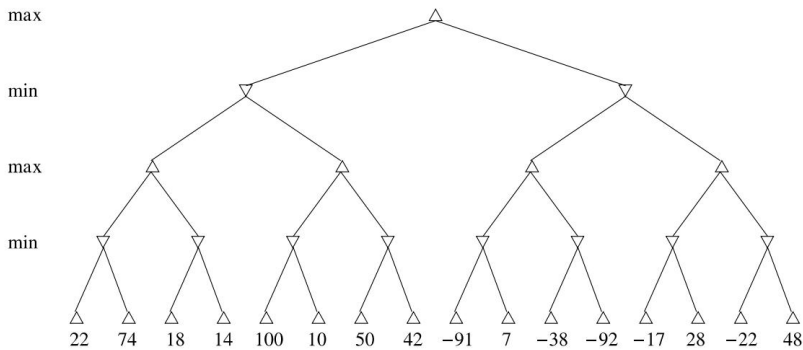


Abb. 6.21 Minimax-Suchbaum

(b) Zeigen Sie, dass dies für unterschiedliche Kosten nicht der Fall ist.

Übung 6.8 Suchen Sie mithilfe der A*-Suche für das 8-Rätsel (manuell) nach einem Pfad von

der Startknoten

1		3
4	2	6
7	5	8

zum Zielknoten

1	2	3
4	5	6
7	8	

(a) unter Verwendung der Heuristik h1 (Abschn.

6.3.4). (b) unter Verwendung der Heuristik h2 (Abschn. 6.3.4).

Aufgabe 6.9 Konstruieren Sie den A*-Suchbaum für den Stadtgraphen aus Abb. 6.14 auf Seite 97 und verwenden Sie die Flugentfernung nach Ulm als Heuristik. Starten Sie in Bern mit Ulm als Ziel. Achten Sie darauf, dass jede Stadt pro Pfad nur einmal vorkommt.

Übung 6.10 Programm A*-Suche in der Programmiersprache Ihrer Wahl

Verwenden Sie die Heuristiken h1 und h2 und testen Sie diese am 8-Puzzle-Beispiel.

Übung 6.11 Geben Sie eine heuristische Bewertungsfunktion für Zustände an, mit denen HEURIS

TICSEARCH kann als Tiefensuche und als Implementierung einer Breitensuche implementiert werden.

Aufgabe 6.12 Welcher Zusammenhang besteht zwischen dem Bild des Paares in der Schlucht aus Abb. 6.13 auf Seite 96 und zulässigen Heuristiken?

Aufgabe 6.13 Zeigen Sie, dass die Heuristiken h1 und h2 für das 8-Puzzle aus Abschn. 6.3.4 sind zulässig.

Aufgabe 6.14 (a)

Der Suchbaum für ein Zwei-Personen-Spiel ist in Abb. 6.21 mit den Bewertungen aller Blattknoten dargestellt. Verwenden Sie die Minimax-Suche mit β - γ -Beschneidung von links nach rechts.

Streichen Sie alle nicht besuchten Knoten durch und geben Sie für jeden inneren Knoten die optimale Ergebnisbewertung an. Markieren Sie den

gewählten Weg. (b) Testen Sie sich selbst mit dem Applet [Win] von P. Winston.

Argumentieren mit Unsicherheit

7

Wir haben bereits in Abschn. gezeigt. 4 mit dem Tweety-Problem, dass zweiwertige Logik zu Problemen im alltäglichen Denken führt. In diesem Beispiel führen die Aussagen *Tweety ist ein Pinguin*, *Pinguine sind Vögel* und *Alle Vögel können fliegen* zu der (semantisch falschen) Schlussfolgerung, dass *Tweety fliegen kann*. Die Wahrscheinlichkeitstheorie bietet eine Sprache, in der wir die Aussage formalisieren können. *Fast alle Vögel können fliegen* und daraus Rückschlüsse ziehen. Die Wahrscheinlichkeitstheorie ist eine bewährte Methode, die wir hier verwenden können, da die Unsicherheit darüber, ob Vögel fliegen können, gut durch einen Wahrscheinlichkeitswert modelliert werden kann. Wir werden zeigen, dass Aussagen wie „99 % aller Vögel können fliegen“ zusammen mit der Wahrscheinlichkeitslogik zu korrekten Schlussfolgerungen führen.

Das Denken unter Unsicherheit und begrenzten Ressourcen spielt in Alltagssituationen und auch in vielen technischen Anwendungen von KI eine große Rolle. In diesen Bereichen sind heuristische Verfahren sehr wichtig, wie wir bereits in Kap. 6. Beispielsweise nutzen wir heuristische Techniken bei der Parkplatzsuche im Stadtverkehr. Heuristiken allein reichen oft nicht aus, insbesondere wenn bei unvollständigem Wissen eine schnelle Entscheidung erforderlich ist, wie das folgende Beispiel zeigt. Ein Fußgänger überquert die Straße und ein Auto nähert sich schnell. Um einen schweren Unfall zu verhindern, muss der Fußgänger schnell reagieren. Er ist nicht in der Lage, sich um vollständige Informationen über den Zustand der Welt zu kümmern, die er für die in Kap. besprochenen Suchalgorithmen benötigen würde. 6. Er muss daher unter den gegebenen Randbedingungen (wenig Zeit und wenig, möglicherweise unsicheres Wissen) zu einer optimalen Entscheidung kommen. Wenn er zu lange nachdenkt, wird es gefährlich. In dieser und vielen ähnlichen Situationen (siehe Abb. 7.1 auf Seite 114) ist eine Methode zum Schlussfolgern mit unsicherem und unvollständigem Wissen erforderlich.

Wir wollen die verschiedenen Möglichkeiten des Denkens unter Unsicherheit anhand eines einfachen medizinischen Diagnosebeispiels untersuchen. Wenn ein Patient Schmerzen im rechten Unterbauch und eine erhöhte Anzahl weißer Blutkörperchen (Leukozyten) verspürt, liegt der Verdacht nahe, dass es sich um eine Blinddarmentzündung handeln könnte. Wir modellieren diese Beziehung mithilfe der Aussagenlogik mit der Formel

Magenschmerzen rechts unten $\rightarrow$ Leukozyten $> 10000 \rightarrow$ Blinddarmentzündung



Abb. 7.1 „Lehnen wir uns einfach zurück und überlegen, was zu tun ist!“

Wenn wir das dann wissen

Magenschmerzen rechts unten $\ddot{y}$ Leukozyten > 10000

wahr ist, dann können wir den Modus Ponens verwenden, um *eine Appendizitis abzuleiten*. Dieses Modell ist eindeutig zu grob. 1976 erkannten Shortliffe und Buchanan dies, als sie ihr medizinisches Expertensystem MYCIN bauten [Sho76]. Sie entwickelten einen Kalkül mithilfe sogenannter Gewissheitsfaktoren, der es ermöglichte, die Gewissheit von Fakten und Regeln darzustellen.

Einer Regel $A \ddot{y} B$ wird ein Sicherheitsfaktor $\ddot{y}$ zugewiesen. Die Semantik einer Regel $A \ddot{y} B$ wird über die bedingte Wahrscheinlichkeit $P(B|A) = \ddot{y}$ definiert. Im obigen Beispiel könnte die Regel dann lauten

Magenschmerzen rechts unten $\ddot{y}$ Leukozyten $> 10000 \ddot{y} 0,6$ Blinddarmentzündung.

Um mit dieser Art von Formeln zu argumentieren, verwendeten sie eine Infinitesimalrechnung, um die Faktoren von Regeln zu verbinden. Es stellte sich jedoch heraus, dass mit dieser Rechnung inkonsistente Ergebnisse abgeleitet werden konnten.

Wie in Kap. besprochen, 4 gab es auch Versuche, dieses Problem durch den Einsatz nicht-monotoner Logik und Default-Logik zu lösen, die jedoch letztlich erfolglos blieben. Die Dempster-Schäfer-Theorie weist einem logischen Satz A eine Glaubensfunktion $\text{Bel}(A)$ zu, deren Wert den Beweisgrad für die Wahrheit von A angibt. Aber auch dieser Formalismus weist Schwächen auf, was in [Pea88] anhand einer Variante von gezeigt wird das Tweety-Beispiel. Selbst die Fuzzy-Logik, die vor allem in der Kontrolltheorie erfolgreich ist, weist bei der Argumentation unter Unsicherheit in komplexeren Anwendungen erhebliche Schwächen auf [Elk93].

Seit etwa Mitte der 1980er Jahre hat die Wahrscheinlichkeitstheorie immer mehr Einfluss auf die KI gehabt [Pea88, Che85, Whi96, Jen01]. Im Bereich des Denkens mit Bayes'schen Netzen bzw. der subjektiven Wahrscheinlichkeit hat es sich einen festen Platz unter den erfolgreichen KI-Techniken gesichert. Anstelle der in der Logik bekannten Implikation (materielle Implikation) wird hier die bedingte Wahrscheinlichkeit verwendet, die das alltägliche Kausaldenken deutlich besser modelliert. Das Schließen mit Wahrscheinlichkeiten profitiert stark von der Tatsache, dass die Wahrscheinlichkeitstheorie ein Hunderte Jahre alter, gut etablierter Zweig der Mathematik ist.

In diesem Kapitel wählen wir einen eleganten, aber für ein Lehrbuch etwas ungewöhnlichen Einstiegspunkt in dieses Gebiet. Nach einer kurzen Einführung in die wichtigsten Grundlagen, die hier für das Schließen mit Wahrscheinlichkeit benötigt werden, beginnen wir mit einem einfachen, aber wichtigen Beispiel für das Schließen mit unsicherem und unvollständigem Wissen. Auf ganz natürliche, fast zwingende Weise werden wir zur Methode der maximalen Entropie (MaxEnt) geführt. Anschließend zeigen wir den Nutzen dieser Methode in der Praxis anhand des medizinischen Expertensystems LEXYMED. Abschließend stellen wir die inzwischen weit verbreitete Argumentation mit Bayes'schen Netzen vor und zeigen die Beziehung zwischen den beiden Methoden.

7.1 Rechnen mit Wahrscheinlichkeiten

Der Leser, der mit der Wahrscheinlichkeitstheorie vertraut ist, kann diesen Abschnitt überspringen. Für alle anderen geben wir einen kurzen Überblick und empfehlen ein paar geeignete Lehrbücher wie [Ros09, FPP07].

Die Wahrscheinlichkeit eignet sich besonders gut zur Modellierung des Denkens unter Unsicherheit. Ein Grund dafür ist, dass Wahrscheinlichkeiten intuitiv leicht zu interpretieren sind, was im folgenden elementaren Beispiel zu sehen ist.

Beispiel 7.1 Bei einem einzelnen Würfelwurf (Experiment) beträgt die Wahrscheinlichkeit des Ereignisses „eine Sechs würfeln“ $1/6$, während die Wahrscheinlichkeit des Ereignisses „eine ungerade Zahl würfeln“ $1/2$ beträgt.

Definition 7.1 Sei Ω die endliche Menge von *Ereignissen* für ein Experiment. Jedes Ereignis $\omega \in \Omega$ repräsentiert ein mögliches Ergebnis des Wurfs. Wenn sich diese Ereignisse ω_i gegenseitig ausschließen, aber alle möglichen Ergebnisse des Versuchs abdecken, werden sie *Elementarereignisse* genannt.

Beispiel 7.2 Für einen einzelnen Wurf eines Spielwürfels

$$\tilde{y} = \{1, 2, 3, 4, 5, 6\}$$

weil keine zwei dieser Ereignisse gleichzeitig passieren können. Eine gerade Zahl würfeln ($\{2, 4, 6\}$) ist daher kein Elementarereignis und auch kein Würfeln einer Zahl kleiner als fünf ($\{1, 2, 3, 4\}$), weil $\{2, 4, 6\} \cap \{1, 2, 3, 4\} = \{2, 4\} = \tilde{y}$.

Gegeben zwei Ereignisse A und B, ist $A \tilde{\cap} B$ ebenfalls ein Ereignis. $\tilde{y}$ selbst wird als das **Gewisse** bezeichnet **Ereignis** und die leere Menge $\emptyset$ das **unmögliche Ereignis**.

Im Folgenden verwenden wir die aussagenlogische Notation für Mengenoperationen. Das heißt, für die Menge $A \tilde{\cap} B$ schreiben wir $A \tilde{\wedge} B$. Dies ist nicht nur eine syntaktische Transformation, Vielmehr ist es auch semantisch korrekt, da der Schnittpunkt zweier Mengen definiert ist als

$$x \tilde{\wedge} A \tilde{\wedge} B \tilde{\wedge} x \tilde{\wedge} A \tilde{\wedge} x \tilde{\wedge} B.$$

Da dies die Semantik von $A \tilde{\wedge} B$ ist, können und werden wir diese Notation verwenden. Das ist auch wahr für die anderen Mengenoperationen Vereinigung und Komplement, und wir werden es tun, wie in gezeigt In der folgenden Tabelle verwenden Sie auch für sie die aussagenlogische Notation.

Mengennotation	Aussagenlogik	Beschreibung
$A \tilde{\cap} B$	$A \tilde{\wedge} B$	Kreuzung/und
$A \tilde{\cup} B$	$A \tilde{\vee} B$	Union/oder
A	$\neg A$	Ergänzung/Negation
$\tilde{y}$	w	Bestimmtes Ereignis/wahr
$\emptyset$	F	Unmögliches Ereignis/falsch

Die hier verwendeten Variablen (zum Beispiel A, B usw.) werden **Zufallsvariablen** genannt in der Wahrscheinlichkeitstheorie. Wir werden hier nur diskrete Zufallsvariablen mit endlichen Grundfunktionen verwenden. Die Variable *face_number* für einen Würfelwurf ist diskret mit den Werten 1, 2, 3, 4, 5, 6. Die Wahrscheinlichkeit, eine Fünf oder Sechs zu würfeln, beträgt 1/3. Das kann sein beschrieben von

$$P(\text{Gesichtsnummer} \tilde{\wedge} \{5, 6\}) = P(\text{Gesichtsnummer} = 5 \tilde{\vee} \text{Gesichtsnummer} = 6) = 1/3.$$

Der Begriff der Wahrscheinlichkeit soll uns eine möglichst objektive Beschreibung geben möglicher Teil unseres „Glaubens“ oder unserer „Überzeugung“ über das Ergebnis eines Experiments. Alle Zahlen im Intervall $[0, 1]$ sollten möglich sein, wobei 0 die Wahrscheinlichkeit dafür ist unmögliches Ereignis und 1 die Wahrscheinlichkeit des bestimmten Ereignisses. Wir kommen hierher die folgende Definition.

Definition 7.2 Sei $\tilde{y} = \{\tilde{y}_1, \tilde{y}_2, \dots, \tilde{y}_n\}$ endlich. Es gibt kein bevorzugtes Elementarereignis, was bedeutet, dass wir eine Symmetrie annehmen, die mit der Häufigkeit des Auftretens jedes Elementarereignisses zusammenhängt. Die *Wahrscheinlichkeit* $P(A)$ des Ereignisses A beträgt dann

$$P(A) = \frac{|A|}{|\tilde{y}|} = \frac{\text{Anzahl günstiger Fälle für } A}{\text{Anzahl möglicher Fälle}}.$$

Daraus folgt sofort, dass jedes Elementarereignis die Wahrscheinlichkeit $1/|\tilde{y}|$ hat. Die Forderung, dass Elementarereignisse die gleiche Wahrscheinlichkeit haben, wird *Laplace-Annahme* genannt und die dadurch berechneten Wahrscheinlichkeiten werden *Laplace-Wahrscheinlichkeiten* genannt. Diese Definition stößt an ihre Grenzen, wenn die Zahl der Elementarereignisse unendlich wird. Da wir hier jedoch nur endliche Ereignisräume betrachten, stellt dies kein Problem dar. Zur Beschreibung von Ereignissen verwenden wir Variablen mit der entsprechenden Anzahl an Werten. Beispielsweise kann eine Variable *eye_color* die Werte *grün, blau, braun* annehmen. *eye_color = blue* beschreibt dann ein Ereignis, da es sich um einen Satz mit den Wahrheitswerten *true/false* (booleschen) Variablen ist die Variable selbst bereits ein Satz. Hier genügt es beispielsweise, $P(\text{JohnCalls})$ statt $P(\text{JohnCalls} = t)$ zu schreiben.

Beispiel 7.3 Nach dieser Definition beträgt die Wahrscheinlichkeit, eine gerade Zahl zu würfeln

$$P(\text{face\_number} \in \{2, 4, 6\}) = \frac{|\{2, 4, 6\}|}{|\{1, 2, 3, 4, 5, 6\}|} = \frac{3}{6} = \frac{1}{2}.$$

Die folgenden wichtigen Regeln ergeben sich direkt aus der Definition.

Satz 7.1 1. P

$(\tilde{y}) = 1$.

2. $P(\emptyset) = 0$, was bedeutet, dass das unmögliche Ereignis eine Wahrscheinlichkeit von 0 hat.

3. Für paarweise exklusive Ereignisse A und B gilt $P(A \cup B) = P(A) + P(B)$.

4. Für zwei komplementäre Ereignisse A und $\neg A$ gilt $P(A) + P(\neg A) = 1$.

5. Für beliebige Ereignisse A und B gilt $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

6. Für $A \cap B$ gilt $P(A \cap B) \leq P(A)$ und $P(B)$.

7. Wenn $A_1, \dots, A_n$ Elementarereignisse sind, dann Bedingung) $\sum_{i=1}^n P(A_i) = 1$ (Normalisierung)

Der Ausdruck $P(A \dot{\vee} B)$ oder äquivalent $P(A, B)$ steht für die Wahrscheinlichkeit der Ereignisse $A \dot{\vee} B$. Wir sind oft an den Wahrscheinlichkeiten aller Elementarereignisse interessiert Ereignisse, also aller Kombinationen aller Werte der Variablen A und B. Für die binäre Variablen A und B sind dies $P(A, B)$, $P(A, \neg B)$, $P(\neg A, B)$, $P(\neg A, \neg B)$. Wir nennen den Vektor

$$(P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

bestehend aus diesen vier Werten eine *Verteilung* oder *gemeinsame Wahrscheinlichkeitsverteilung* der Variablen A und B. Eine Kurzform hierfür ist $P(A, B)$. Die Verteilung im Fall von Zwei Variablen können gut in Form einer Tabelle (Matrix) visualisiert werden, dargestellt als folgt:

	$P(A, B) \quad B = w$	$B = f$
$A = w$	$P(A, B)$	$P(A, \neg B)$
$A = f$	$P(\neg A, B)$	$P(\neg A, \neg B)$

Für die d Variablen $X_1, \dots, X_d$ mit jeweils n Werten hat die Verteilung die Werte

$P(X_1 = x_1, \dots, X_d = x_d)$ und $x_1, \dots, x_d$, die jeweils n verschiedene Werte annehmen.

Die Verteilung kann daher als d-dimensionale Matrix mit einer Summe dargestellt werden

von n^d Elemente. Aufgrund der Normalisierungsbedingung aus Satz 7.1 auf Seite 117 gilt

jedoch einer dieser n^d Die Werte sind redundant und die Verteilung ist charakterisiert von $n^d - 1$ eindeutige Werte.

7.1.1 Bedingte Wahrscheinlichkeit

Beispiel 7.4 Auf der Landsdowne Street in Boston wird die Geschwindigkeit von 100 Fahrzeugen gemessen. Bei jeder Messung wird auch vermerkt, ob es sich beim Fahrer um einen Schüler handelt. Der Ergebnisse sind

Ereignis	Frequenz	Relative Frequenz
Fahrzeug beobachtet	100	1
Fahrer ist Student (S)	30	0,3
Geschwindigkeit zu hoch (G)	10	0,1
Der Fahrer ist Student und fährt zu schnell (S $\dot{\vee}$ G)	5	0,05

Wir stellen die Frage: *Beschleunigen Schüler häufiger als der Durchschnitt? Sohn, oder als Nicht-Studenten?*¹

Die Antwort ergibt sich aus der Wahrscheinlichkeit

$$P(G|S) = \frac{| \text{Fahrer ist Student und fährt zu schnell} |}{| \text{Fahrer ist Student} |} = \frac{5}{30} = \frac{1}{6} \dot{\vee} 0,17$$

¹Die berechneten Wahrscheinlichkeiten können nur dann für fortgesetzte Aussagen verwendet werden, wenn die gemessene Stichprobe vorliegt (100 Fahrzeuge) ist repräsentativ. Ansonsten können nur Aussagen zu den beobachteten 100 Fahrzeugen gemacht werden gemacht sein.

wegen Geschwindigkeitsüberschreitung, sofern der Fahrer ein Student ist. Dies unterscheidet sich offensichtlich von der A-priori-Wahrscheinlichkeit $P(G) = 0,1$ für Geschwindigkeitsüberschreitungen. Für die A-priori-Wahrscheinlichkeit ist der Ereignisraum nicht durch zusätzliche Bedingungen begrenzt.

Definition 7.3 Für zwei Ereignisse A und B ist die Wahrscheinlichkeit $P(A|B)$ für A unter der Bedingung B (*bedingte Wahrscheinlichkeit*) definiert durch

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

In Beispiel 7.4 sehen wir, dass im Fall eines endlichen Ereignisraums die bedingte Wahrscheinlichkeit $P(A|B)$ als die Wahrscheinlichkeit von $A \cap B$ verstanden werden kann, wenn wir nur das Ereignis B betrachten, also als

$$P(A|B) = \frac{|A \cap B|}{|B|}.$$

Diese Formel kann leicht mithilfe von Definition 7.2 auf Seite 117 abgeleitet werden

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{|A \cap B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{|A \cap B|}{|B|}.$$

Definition 7.4 Wenn für zwei Ereignisse A und B

$$P(A|B) = P(A),$$

dann heißen diese Ereignisse unabhängig.

Somit sind A und B unabhängig, wenn die Wahrscheinlichkeit des Ereignisses A nicht durch das Ereignis B beeinflusst wird.

Satz 7.2 Für unabhängige Ereignisse A und B folgt aus der Definition, dass

$$P(A \cap B) = P(A) \cdot P(B).$$

Beispiel 7.5 Bei einem Wurf mit zwei Würfeln beträgt die Wahrscheinlichkeit, zwei Sechsen zu würfeln, $1/36$, wenn die beiden Würfel unabhängig sind, weil

$$P(D1 = 6 \text{ \& } D2 = 6) = P(D1 = 6) \cdot P(D2 = 6) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36},$$

wobei die erste Gleichung nur dann wahr ist, wenn die beiden Würfel unabhängig sind. Wenn zum Beispiel durch eine magische Kraft Würfel 2 immer gleich Würfel 1 ist, dann

$$P(D1 = 6 \text{ \& } D2 = 6) = \frac{1}{6}.$$

Kettenregel

Das Lösen der Definition der bedingten Wahrscheinlichkeit für $P(A|B)$ führt zur sogenannten Produktregel

$$P(A \text{ \& } B) = P(A|B)P(B),$$

was wir sofort für den Fall von n Variablen verallgemeinern. Durch wiederholte Anwendung der obigen Regel erhalten wir die *Kettenregel*

$$\begin{aligned} P(X_1, \dots, X_n) &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_1, \dots, X_{n-1}) \\ &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_{n-1} | X_1, \dots, X_{n-2}) \cdot P(X_1, \dots, X_{n-2}) \\ &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_{n-1} | X_1, \dots, X_{n-2}) \cdot \dots \cdot P(X_2 | X_1) \cdot P(X_1) \\ &= \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}), \end{aligned} \quad (7.1)$$

mit dem wir eine Verteilung als Produkt bedingter Wahrscheinlichkeiten darstellen können. Da die Kettenregel für alle Werte der Variablen $X_1, \dots, X_n$ gilt, wurde sie für die Verteilung mit dem Symbol P formuliert.

Marginalisierung

Weil $A \text{ \& } (A \text{ \& } B) \text{ \& } (A \text{ \& } \neg B)$ für binäre Variablen A und B gilt

$$P(A) = P((A \text{ \& } B) \text{ \& } (A \text{ \& } \neg B)) = P(A \text{ \& } B) + P(A \text{ \& } \neg B).$$

Durch Summation über die beiden Werte von B wird die Variable B eliminiert. Analog kann für beliebige Variablen $X_1, \dots, X_d$ eine Variable, beispielsweise X_d , durch Summation über alle ihre Variablen eliminiert werden, und wir erhalten

$$P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}) = \sum_{x_d} P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}, X_d = x_d).$$

Die Anwendung dieser Formel nennt man **Marginalisierung**. Diese Summation kann mit den Variablen $X_1, \dots, X_{d-1}$ fortgesetzt werden, bis nur noch eine Variable übrig ist. **Marginalisierung** kann auch auf die Verteilung $P(X_1, \dots, X_d)$ angewendet werden. Die resultierende Verteilung $P(X_1, \dots, X_{d-1})$ heißt **Randverteilung**. Es ist vergleichbar mit der Projektion eines rechteckigen Quaders auf eine ebene Fläche. Hier ist das dreidimensionale Objekt auf der Kante oder dem „Rand“ des Quaders gezeichnet, also auf einer zweidimensionalen Menge. Sowohl in diesen Fällen wird die Dimensionalität um eins reduziert.

Beispiel 7.6 Wir beobachten die Menge aller Patienten, die akut zum Arzt kommen Magenschmerzen. Für jeden Patienten wird der Leukozytenwert gemessen, der ein Maß darstellt für die relative Häufigkeit weißer Blutkörperchen im Blut. Wir definieren die Variable *Leuko*, was genau dann zutrifft, wenn der Leukozytenwert größer als 10.000 ist. Das weist auf eine Infektion im Körper hin. Ansonsten definieren wir die Variable *App*, die erzählt Wir informieren Sie, ob der Patient an einer Blinddarmentzündung, also einem entzündeten Blinddarm, leidet. Die Verteilung $P(\text{App}, \text{Leuko})$ dieser beiden Variablen ist in der folgenden Tabelle angegeben:

$P(\text{App}, \text{Leuko})$	App	$\neg \text{App}$	Total
Leuko	0,23	0,31	0,54
$\neg \text{Leuko}$	0,05	0,41	0,46
Gesamt	0,28	0,72	1

In der letzten Zeile ist die Summe über die Zeilen angegeben und in der letzten Spalte die Summe der Spalten ist angegeben. Zu diesen Beträgen kommt es durch Marginalisierung. Zum Beispiel wir ablesen

$$P(\text{Leuko}) = P(\text{App}, \text{Leuko}) + P(\neg \text{App}, \text{Leuko}) = 0,54.$$

Die angegebene Verteilung $P(\text{App}, \text{Leuko})$ könnte aus einer Befragung deutscher Ärzte stammen, Zum Beispiel. Daraus können wir dann die bedingte Wahrscheinlichkeit berechnen

$$P(\text{Leuko}|\text{App}) = \frac{P(\text{Leuko}, \text{App})}{P(\text{App})} = 0,82$$

Daraus lässt sich schließen, dass etwa 82 % aller Blinddarmentzündungen zu einem hohen Leukozytenwert führen. Solche Werte werden in der medizinischen Fachliteratur veröffentlicht. Allerdings wäre die bedingte Wahrscheinlichkeit $P(\text{App}|\text{Leuko})$ eigentlich viel hilfreicher für die Diagnose Blinddarmentzündung, wird nicht veröffentlicht. Um dies zu verstehen, leiten wir zunächst ein einfaches Aber ab sehr wichtige Formel.

Satz von Bayes

Das Vertauschen von A und B in Definition 7.3 auf Seite 119 führt zu

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \text{und} \quad P(B|A) = \frac{P(A \cap B)}{P(A)}.$$

Indem wir die beiden Gleichungen nach $P(A|B)$ lösen und gleichsetzen, erhalten wir den **Satz von Bayes**

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}, \quad (7.2)$$

was wir sofort auf das Blinddarmentzündungsproblem anwenden und erhalten

$$P(App|Leuko) = \frac{P(Leuko|App) \cdot P(App)}{P(Leuko)} = \frac{0,82 \cdot 0,28}{0,54} = 0,43. \quad (7.3)$$

Warum wird nun $P(Leuko|App)$ veröffentlicht, aber nicht $P(App|Leuko)$?

Unter der Annahme, dass Blinddarmentzündung den menschlichen Körper unabhängig von der Rasse betrifft, ist $P(Leuko|App)$ ein universeller Wert, der auf der ganzen Welt gilt. In (7.3) sehen wir, dass $P(App|Leuko)$ nicht universell ist, da dieser Wert von den A-priori-Wahrscheinlichkeiten $P(App)$ und $P(Leuko)$ beeinflusst wird. Jedes davon kann je nach Lebensumständen variieren. Beispielsweise hängt $P(Leuko)$ davon ab, ob es in einer Population viele oder wenige Infektionskrankheiten gibt. Dieser Wert könnte zwischen den Tropen und kälteren Regionen möglicherweise erheblich variieren. Der Satz von Bayes ermöglicht jedoch eine einfache Berechnung des für die Diagnose relevanten $P(App|Leuko)$ aus dem allgemeingültigen Wert $P(Leuko|App)$.

Bevor wir näher auf dieses Beispiel eingehen und es zu einem medizinischen Expertensystem für Blinddarmentzündung erweitern, müssen wir zunächst die notwendige probabilistische Schlussfolgerung einführen. Mechanismus.

7.2 Das Prinzip der maximalen Entropie

Wir werden nun anhand eines Inferenzbeispiels zeigen, dass ein Kalkül zum Schließen unter Unsicherheit mithilfe der Wahrscheinlichkeitstheorie realisiert werden kann. Allerdings werden wir bald sehen, dass die eingefahrenen probabilistischen Pfade schnell ein Ende haben. Insbesondere wenn zu wenig Wissen vorhanden ist, um die notwendigen Gleichungen zu lösen, sind neue Ideen gefragt. Der amerikanische Physiker ET Jaynes leistete in den 1950er Jahren Pionierarbeit auf diesem Gebiet. Er behauptete, dass man bei fehlendem Wissen die Entropie der gewünschten Wahrscheinlichkeitsverteilung maximieren kann, und wandte dieses Prinzip auf viele Beispiele in [Jay57, Jay03] an. Dieses Prinzip wurde dann weiterentwickelt [Che83, Nil86, Kan89, KK92] und ist mittlerweile ausgereift und technologisch anwendbar, was wir am Beispiel des LEXMED-Projekts in Abschn. 7.3.

7.2.1 Eine Inferenzregel für Wahrscheinlichkeiten

Wir wollen eine zum Modus Ponens analoge Inferenzregel für unsicheres Wissen ableiten. Aus der Kenntnis eines Satzes A und einer Regel $A \rightarrow B$ soll die Schlussfolgerung B gezogen werden. Kurz und bündig formuliert lautet das:

$$\frac{A, A \rightarrow B}{B}.$$

Die Verallgemeinerung für Wahrscheinlichkeitsregeln ergibt

$$\underline{P(A) = \tilde{y}, P(B|A) = \tilde{y} \quad P(B) = ?}.$$

Gegeben seien die beiden Wahrscheinlichkeitsregeln $\tilde{y}$, $\tilde{y}$ und der gewünschte Wert $P(B)$. Durch Marginalisierung erhalten wir die gewünschte Randverteilung

$$P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A).$$

Die drei Werte $P(A)$, $P(\neg A)$, $P(B|A)$ auf der rechten Seite sind bekannt, der Wert $P(B|\neg A)$ ist jedoch unbekannt. Über $P(B)$ können wir mit der klassischen Wahrscheinlichkeitstheorie keine exakte Aussage treffen, aber wir können allenfalls $P(B) \approx P(B|A) \cdot P(A)$ abschätzen.

Wir betrachten nun die Verteilung

$$P(A, B) = (P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

und stellen Sie stenografisch die vier Unbekannten vor

$$\begin{aligned} p_1 &= P(A, B), p_2 \\ &= P(A, \neg B), p_3 = P \\ &(\neg A, B), \\ p_4 &= P(\neg A, \neg B). \end{aligned}$$

Diese vier Parameter bestimmen die Verteilung. Wenn sie alle bekannt sind, kann jede Wahrscheinlichkeit für die beiden Variablen A und B berechnet werden. Um die vier Unbekannten zu berechnen, werden vier Gleichungen benötigt. Eine Gleichung ist in Form der Normalisierungsbedingung bereits bekannt

$$p_1 + p_2 + p_3 + p_4 = 1.$$

Daher sind drei weitere Gleichungen erforderlich. In unserem Beispiel sind jedoch nur zwei Gleichungen bekannt.

Aus den gegebenen Werten $P(A) = \tilde{y}$ und $P(B|A) = \tilde{y}$ berechnen wir

$$P(A, B) = P(B|A) \cdot P(A) = \tilde{y}\tilde{y}$$

Und

$$P(A) = P(A, B) + P(A, \neg B).$$

Daraus können wir das folgende Gleichungssystem aufstellen und soweit wie möglich lösen:

$$p_1 = \bar{y}\bar{y}, \quad (7.4)$$

$$p_1 + p_2 = \bar{y}, \quad (7.5)$$

$$p_1 + p_2 + p_3 + p_4 = 1, \quad (7.6)$$

$$(7.4) \text{ in } (7.5): \quad p_2 = \bar{y} \bar{y} \bar{y} = \bar{y}(1 \bar{y} \bar{y}), \quad (7.7)$$

$$(7.5) \text{ in } (7.6): \quad p_3 + p_4 = 1 \bar{y} \bar{y}. \quad (7.8)$$

Damit sind die Wahrscheinlichkeiten p_1, p_2 für die Interpretationen (A, B) und $(A, \neg B)$ bekannt, für die Werte p_3, p_4 bleibt jedoch nur noch eine Gleichung übrig. Um trotz dieses fehlenden Wissens zu einer eindeutigen Lösung zu kommen, ändern wir unseren

Standpunkt. Wir verwenden die gegebene Gleichung als Randbedingung für die Lösung eines Optimierungsproblems. Gesucht wird eine Verteilung p (für die Variablen p_3, p_4), die die Entropie maximiert

$$H(p) = - \sum_{i=1}^n p_i \ln p_i = -p_3 \ln p_3 - p_4 \ln p_4 \quad (7.9)$$

unter der Nebenbedingung $p_3 + p_4 = 1 \bar{y} \bar{y}$ (7.8). Warum genau sollte die Entropiefunktion

maximiert werden? Da uns Informationen über die Verteilung fehlen, müssen diese

irgendwie hinzugefügt werden. Wir könnten einen Ad-hoc-Wert festlegen, zum Beispiel

$p_3 = 0,1$. Es ist jedoch besser, die Werte p_3 und p_4 so zu bestimmen, dass die hinzugefügte Information

Wir können zeigen (Abschn. 8.4.2 und [SW76]), dass die Entropie die Unsicherheit einer

Verteilung bis zu einem konstanten Faktor misst. Die negative Entropie ist dann ein

Maß für die Menge an Informationen, die eine Verteilung enthält. Durch die Maximierung

der Entropie wird der Informationsgehalt der Verteilung minimiert. Um dies zu

veranschaulichen, wird die Entropiefunktion für den zweidimensionalen Fall in Abb. 7.2 auf Seite 125

Um das Maximum der Entropie unter der Nebenbedingung $p_3 + p_4 = 1 \bar{y} \bar{y} = 0$ zu

bestimmen, verwenden wir die Methode der Lagrange-Multiplikatoren [Ste07]. Die Lagrange-Funktion lautet

$$L = -p_3 \ln p_3 - p_4 \ln p_4 + \bar{y}(p_3 + p_4 - 1 \bar{y} \bar{y}).$$

Durch die partiellen Ableitungen nach p_3 und p_4 erhalten wir:

$$\begin{aligned} \frac{\partial L}{\partial p_3} &= -\ln p_3 - 1 + \bar{y} = 0, \quad \bar{y} p_3 \\ &= \\ -\ln p_4 - 1 + \bar{y} &= 0 \quad \bar{y} p_4 \end{aligned}$$

Abb. 7.2

**Höhenliniendiagramm
der zweidimensionalen**

**Entropiefunktion. Wir sehen,
dass es im gesamten**

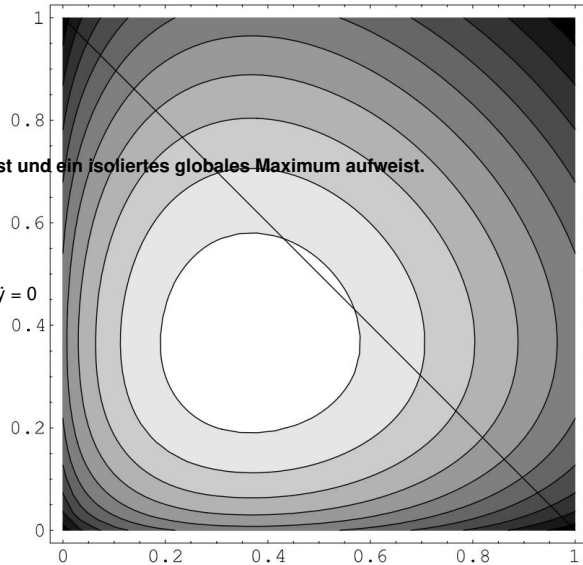
Einheitsquadrat streng konvex ist und ein isoliertes globales Maximum aufweist.

Ebenfalls markiert ist die

**Nebenbedingung $p_3 + p_4 = 1$
als Sonderfall**

der hier relevanten

Bedingung $p_3 + p_4 \ddot{y} 1 + \ddot{y} = 0$ für $\ddot{y} = 0$



und berechnen

$$p_3 = p_4 = \frac{1 - \ddot{y}}{2}.$$

Jetzt können wir den gewünschten Wert berechnen

$$P(B) = P(A, B) + P(\neg A, B) = p_1 + p_3 = \ddot{y} + \frac{1 - \ddot{y}}{2} = \frac{1 + \ddot{y}}{2}.$$

Das Einsetzen von $\ddot{y}$ und $\ddot{y}$ ergibt

$$P(B) = P(A) P(B|A) \ddot{y} \frac{1}{2} = \frac{1}{2} \ddot{y}.$$

$P(B)$ ist in Abb. 7.3 auf Seite 126 für verschiedene Werte von $P(B|A)$ dargestellt. Wir sehen, dass im Fall der zweiwertigen Kante, das heißt, wenn $P(B)$ und $P(B|A)$ die Werte 0 oder 1 annehmen, die probabilistische Inferenz denselben Wert für $P(B)$ zurückgibt wie der Modus Ponens. Wenn A und $B|A$ beide wahr sind, ist auch B wahr. Ein interessanter Fall ist $P(A) = 0$, in dem $\neg A$ wahr ist. Modus ponens kann hier nicht angewendet werden, aber unsere Formel ergibt den Wert $1/2$ für $P(B)$, unabhängig von $P(B|A)$. Wenn A falsch ist, wissen wir nichts über B , was genau unsere Intuition widerspiegelt. Der Fall, dass $P(A) = 1$ und $P(B|A) = 0$ ist, wird auch von der Aussagenlogik abgedeckt. Hier ist A wahr und $A \ddot{y} B$ falsch und somit $A \ddot{y} \neg B$ wahr. Dann ist B falsch. Die horizontale Linie in der Abbildung bedeutet, dass wir im Fall von $P(B|A) = 1/2$ keine Vorhersage über B machen können. Zwischen diesen Punkten ändert sich $P(B)$ linear bei Änderungen von $P(A)$ oder $P(B|A)$.

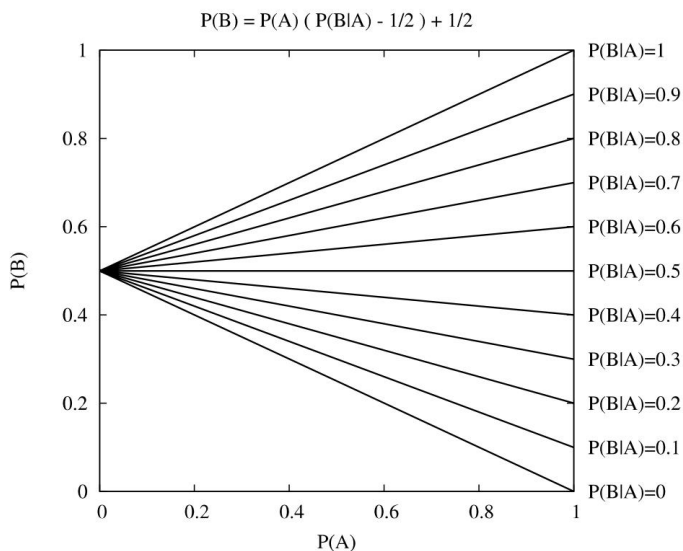


Abb. 7.3 Kurvenfeld für $P(B)$ als Funktion von $P(A)$ für verschiedene Werte von $P(B|A)$

Satz 7.3 *Es gebe einen konsistenten² Satz linearer Wahrscheinlichkeitsgleichungen. Dann gibt es ein eindeutiges Maximum für die Entropiefunktion mit den gegebenen Gleichungen als Randbedingungen. Die dadurch definierte MaxEnt-Verteilung hat unter den Randbedingungen einen minimalen Informationsgehalt.*

Aus diesem Satz folgt, dass es keine Verteilung gibt, die die Einschränkungen erfüllt und eine höhere Entropie aufweist als die MaxEnt-Verteilung. Eine Rechnung, die zu einer geringeren Entropie führt, fügt zusätzliche Ad-hoc-Informationen ein, die nicht gerechtfertigt sind.

Wenn wir die obige Berechnung von $P(B)$ genauer betrachten, sehen wir, dass die beiden Werte p_3 und p_4 immer symmetrisch auftreten. Das bedeutet, dass sich durch den Austausch der beiden Variablen das Ergebnis nicht ändert. Somit ist das Endergebnis $p_3 = p_4$. Die sogenannte Indifferenz dieser beiden Variablen führt dazu, dass sie von MaxEnt gleichgesetzt werden. Dieser Zusammenhang gilt allgemein:

Definition 7.5 Wenn ein beliebiger Austausch von zwei oder mehr Variablen in den La-Grange-Gleichungen zu äquivalenten Gleichungen führt, werden diese Variablen als *indifferent bezeichnet*.

²Ein Satz probabilistischer Gleichungen heißt konsistent, wenn es mindestens eine Lösung gibt, also eine Verteilung, die alle Gleichungen erfüllt.

Satz 7.4 Wenn eine Menge von Variablen $\{p_1, \dots, p_k\}$ indifferent ist, dann liegt das Maximum der Entropie unter den gegebenen Einschränkungen an dem Punkt, an dem $p_1 = p_2 = \dots = p_k$.

Mit diesem Wissen hätten wir die beiden Variablen p_3 und p_4 sofort gleichsetzen können (ohne die Lagrange-Gleichungen zu lösen).

7.2.2 Maximale Entropie ohne explizite Einschränkungen

Wir betrachten nun den Fall, in dem kein Wissen gegeben ist. Dies bedeutet, dass es sich nicht um die Normalisierungsbedingung handelt

$$p_1 + p_2 + \dots + p_n = 1,$$

es gibt keine Einschränkungen. Alle Variablen sind daher indifferent. Deshalb können wir [3](#)

Setze sie gleich und es folgt, dass $p_1 = p_2 = \dots = p_n = 1/n$. Für das Denken unter Unsicherheit bedeutet dies, dass bei völligem Mangel an Wissen alle Welten gleich wahrscheinlich sind. Das heißt, die Verteilung ist gleichmäßig. Im Fall der beiden Variablen A und B wäre dies beispielsweise der Fall

$$P(A, B) = P(A, \neg B) = P(\neg A, B) = P(\neg A, \neg B) = 1/4,$$

woraus $P(A) = P(B) = 1/2$ und $P(B|A) = 1/2$ folgen. Das Ergebnis für den zweidimensionalen Fall ist in Abb. [7.2](#) auf Seite [125](#) zu sehen, da die markierte Bedingung genau die Normalisierungsbedingung ist. Wir sehen, dass das Maximum der Entropie genau bei $(1/2, 1/2)$ auf der Linie liegt.

Sobald der Wert einer Bedingung von dem aus der Gleichverteilung abgeleiteten Wert abweicht, verschieben sich die Wahrscheinlichkeiten der Welten. Wir zeigen dies in einem weiteren Beispiel. Bei den gleichen Beschreibungen wie oben gehen wir nur davon aus

$$P(B|A) = \tilde{y}$$

ist bekannt. Somit ist $P(A, B) = P(B|A)P(A) = \tilde{y}P(A)$, woraus $p_1 = \tilde{y}(p_1 + p_2)$ folgt und wir die beiden Einschränkungen ableiten

$$\tilde{y}p_2 + (\tilde{y} - 1)p_1 = 0,$$

$$p_1 + p_2 + p_3 + p_4 - 1 = 0.$$

³Der Leser kann dieses Ergebnis durch Maximierung der Entropie unter der Normalisierungsbedingung berechnen (Übung [7.5](#) auf Seite [158](#)).

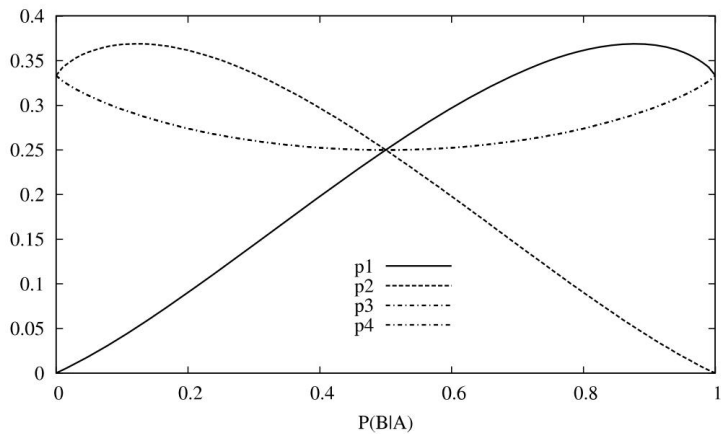


Abb. 7.4 p_1, p_2, p_3, p_4 in Abhängigkeit von $\tilde{y}$

Tabelle 7.1 Wahrheitstabelle für
materielle Implikation und
bedingte Wahrscheinlichkeit für
Grenze der Aussagenlogik

$A \rightarrow B$	$P(A)$	$P(B)$	$P(B A)$
t t t 111			
t f f 100			
f t w f f t 0	0	1	Nicht definiert
		0	Nicht definiert

Hier lassen sich die Lagrange-Gleichungen nicht mehr so einfach symbolisch lösen. Eine numerische Lösung der Lagrange-Gleichungen ergibt das in Abb. 7.4 dargestellte Bild, was zeigt, dass $p_3 = p_4$. Wir können dies bereits an den Einschränkungen erkennen, in denen p_3 und p_4 sind gleichgültig. Für $P(B|A) = 1/2$ erhalten wir die Gleichverteilung, die ist keine Überraschung. Dies bedeutet, dass die Einschränkung für diesen Wert keine Einschränkung der Verteilung mit sich bringt. Darüber hinaus können wir sehen, dass für kleine $P(B|A)$ $P(A, B)$ auch klein.

7.2.3 Bedingte Wahrscheinlichkeit versus materielle Implikation

Wir werden nun zeigen, dass die bedingte Wahrscheinlichkeit für die Modellierung des Denkens besser ist als das, was in der Logik als materielle Implikation bekannt ist (siehe hierzu auch [Ada75]). Zunächst betrachten wir die in Tabelle 7.1 dargestellte Wahrheitstabelle, in der die bedingte Wahrscheinlichkeit und die materielle Implikation für die Extremfälle der Wahrscheinlichkeiten Null und Eins sind verglichen. In beiden Fällen mit falschen Prämissen (die intuitiv kritische Fälle sind) $P(B|A)$ ist undefiniert, was Sinn macht.

Nun fragen wir uns, welchen Wert $P(B|A)$ annimmt, wenn beliebige Werte $P(A) = \bar{y}$ und $P(B) = \bar{y}$ gegeben sind und keine anderen Informationen bekannt sind. Auch hier maximieren wir die Entropie unter den gegebenen Randbedingungen. Wie oben haben wir eingestellt

$$p_1 = P(A, B), p_2 = P(A, \neg B), p_3 = P(\neg A, B), p_4 = P(\neg A, \neg B)$$

und als Nebenbedingungen erhalten

$$p_1 + p_2 = \bar{y}, \quad (7.10)$$

$$p_1 + p_3 = \bar{y}, \quad (7.11)$$

$$p_1 + p_2 + p_3 + p_4 = 1. \quad (7.12)$$

Damit rechnen wir mittels Entropiemaximierung (siehe Übung 7.8 auf Seite 159)

$$p_1 = \bar{y}\bar{y}, p_2 = \bar{y}(1 - \bar{y}), p_3 = \bar{y}(1 - \bar{y}), p_4 = (1 - \bar{y})(1 - \bar{y}).$$

Aus $p_1 = \bar{y}\bar{y}$ folgt $P(A, B) = P(A) \cdot P(B)$, was bedeutet, dass A und B unabhängig sind. Da es keine Einschränkungen zwischen A und B gibt, führt das MaxEnt-Prinzip zur Unabhängigkeit dieser Variablen. Die rechte Hälfte von Tabelle 7.1 auf Seite 128 erleichtert das Verständnis. Aus der Definition

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

Daraus folgt für den Fall $P(A) = 0$, das heißt, wenn die Prämisse nicht falsch ist, weil A und B unabhängig sind, dass $P(B|A) = P(B)$. Für den Fall $P(A) = 0$ bleibt $P(B|A)$ undefiniert.

7.2.4 MaxEnt-Systeme

Wie bereits erwähnt, kann die MaxEnt-Optimierung aufgrund der Nichtlinearität der Entropiefunktion für nichttriviale Probleme normalerweise nicht symbolisch durchgeführt werden. Daher wurden zwei Systeme zur numerischen Entropiemaximierung entwickelt. Das erste System, SPIRIT (Symmetrical Probabilistic Intensional Reasoning in Inference Net Works in Transition, www.xspirit.de), [RM96] wurde an der Fernuniversität Hagen gebaut. Das zweite, PIT (Probability Induction Tool), wurde an der Technischen Universität München entwickelt [Sch96, ES99, SE00]. Wir werden PIT nun kurz vorstellen.

Das PIT-System verwendet die Methode der sequentiellen quadratischen Programmierung (SQP), um ein Extremum der Entropiefunktion unter den gegebenen Einschränkungen zu finden. Als Eingabe erwartet PIT Daten, die die Einschränkungen enthalten. Beispielsweise sind die Nebenbedingungen $P(A) = \bar{y}$ und $P(B|A) = \bar{y}$ aus Abschn. 7.2.1 haben die Form

```

var A{t,f}, B{t,f};

P([A=t]) = 0,6;
P([B=t] | [A=t]) = 0,3;

QP([B=t]);
QP([B=t] | [A=t]);

```

Da PIT eine numerische Berechnung durchführt, müssen wir explizite Wahrscheinlichkeitswerte eingeben. Die vorletzte Zeile enthält die Abfrage $QP([B=t])$. Das heisst dass $P(B)$ der gewünschte Wert ist. Auf www.pit-systems.de unter „Beispiele“ finden Sie jetzt Fügen Sie diese Eingabe in eine leere Eingabeseite („Blank Page“) ein und starten Sie PIT. Als Ergebnis haben wir erhalten

Nr. Wahrheitswert	Wahrscheinlichkeitsabfrage
1 NICHT SPEZIFIZIERT 3.800e-01	$QP([B = t]);$
2 NICHT SPEZIFIZIERT 3.000e-01	$QP([A = t] \rightarrow [B = t]);$

und daraus $P(B) = 0,38$ und $P(B|A) = 0,3$ ablesen.

7.2.5 Das Tweety-Beispiel

Wir zeigen nun anhand des Tweety-Beispiels aus Abschn. 4.3, diese probabilistische Argumentation und insbesondere MaxEnt sind nicht monoton und modellieren das alltägliche Denken sehr Also. Wir modellieren die relevanten Regeln mit Wahrscheinlichkeiten wie folgt:

$P(\text{Vogel} \text{Pinguin}) = 1$	„Pinguine sind Vögel“
$P(\text{Fliegen} \text{Vogel}) \in [0,95, 1]$	„(fast alle) Vögel können fliegen“
$P(\text{Fliegen} \text{Pinguin}) = 0$	„Pinguine können nicht fliegen“

Die erste und dritte Regel stellen feste Vorhersagen dar, die sich auch leicht in der Logik formulieren lassen. Im zweiten bringen wir jedoch unser Wissen zum Ausdruck, dass fast alle Vögel können mithilfe eines Wahrscheinlichkeitsintervalls fliegen. Mit den PIT-Eingabedaten

```

Var-Pinguin{ja, nein}, Vogel{ja, nein}, fliegt{ja, nein};

P([Vogel=Ja] | [Pinguin=Ja]) = 1;
P([flies=yes] | [bird=yes]) IN [0.95,1];
P([flies=yes] | [pinguin=yes]) = 0;

QP([flies=yes] | [pinguin=yes]);

```

Wir bekommen die richtige Antwort zurück

Nr. Wahrheitswert	Wahrscheinlichkeitsabfrage
1	UNSPECIFIED 0.000e+00 QP([penguin=yes]-> [flies=yes]);

mit der These, dass Pinguine nicht fliegen können.⁴ Die Erklärung hierfür ist sehr einfach. Mit $P(\text{Fliegen}|\text{Vogel}) \approx [0,95, 1]$ ist es möglich, dass es nicht fliegende Vögel gibt. Wenn dies würde die Regel durch $P(\text{Fliegen}|\text{Vogel}) = 1$ ersetzt, dann könnte PIT nichts tun und würde eine Fehlermeldung über inkonsistente Einschränkungen ausgeben.

An diesem Beispiel können wir leicht erkennen, dass Wahrscheinlichkeitsintervalle oft sehr hilfreich sind zur Modellierung unserer Unwissenheit über genaue Wahrscheinlichkeitswerte. Wir hätten ein machen können Eine noch unschärfere Formulierung der zweiten Regel im Sinne von „Normalerweise fliegen Vögel“ mit $P(\text{Fliegen}|\text{Vogel}) \approx (0,5, 1]$. Die Verwendung des halboffenen Intervalls schließt den Wert 0,5 aus.

In [Pea88] wurde bereits gezeigt, dass dieses Beispiel auch ohne MaxEnt mit probabilistischer Logik gelöst werden kann. In [Sch96] wird für eine Reihe anspruchsvoller Benchmarks für nichtmonotones Denken gezeigt, dass diese elegant gelöst werden können MaxEnt. Im folgenden Abschnitt stellen wir eine erfolgreiche praktische Anwendung vor MaxEnt in Form eines medizinischen Expertensystems.

7.3 LEXMED, ein Expertensystem zur Diagnose einer Blinddarmentzündung

Das medizinische Expertensystem LEXMED, das die MaxEnt-Methode nutzt, wurde an der Hochschule Ravensburg-Weingarten von Manfred entwickelt Schramm, Walter Rampf und der Autor, in Zusammenarbeit mit dem Weingarten 14-Nothelfer-Krankenhaus [SE00, Le999].⁵ Das Akronym LEXMED steht für *Lernendes Expertensystem für medizinische Diagnose*.

7.3.1 Appendizitis-Diagnose mit formalen Methoden

Die häufigste schwerwiegende Ursache für akute Bauchschmerzen [dD91] ist eine Blinddarmentzündung – eine Entzündung des Blinddarms, eines blinden Schlauchs, der mit dem Blinddarm verbunden ist. Sogar Heutzutage kann die Diagnose in vielen Fällen schwierig sein [OFY+95]. Zum Beispiel bis zu ca 20 % der entfernten Anhänge sind ohne pathologischen Befund, das heißt dass die Operationen unnötig waren. Ebenso gibt es regelmäßig Fälle, in denen Eine Blinddarmentzündung wird nicht als solche erkannt.

Bereits seit Anfang der 1970er Jahre gab es Versuche, die Diagnose einer Appendizitis zu automatisieren, mit dem Ziel, die Rate falscher Diagnosen zu reduzieren [dDLS+72, OPB94, OFY+95]. Besonders hervorzuheben ist das Expertensystem

⁴ $QP([penguin=yes] \rightarrow [flies=yes])$ ist eine alternative Form der PIT-Syntax für $QP([flies=yes] \mid [penguin=yes])$.

⁵Das Projekt wurde vom Land Baden-Württemberg, der Krankenkasse AOK Baden-Württemberg, der Hochschule Ravensburg-Weingarten und finanziert das 14 Nothelfer Krankenhaus in Weingarten.

zur Diagnose akuter Bauchschmerzen, entwickelt von de Dombal in Großbritannien. Es wurde 1972 veröffentlicht, also deutlich früher als das berühmte System MYCIN.

Fast alle formalen diagnostischen Verfahren in der Medizin basieren bisher auf Scores. Punktesysteme sind denkbar einfach anzuwenden: Für jeden Wert eines Symptoms (zum Beispiel *Fieber* oder *Schmerzen im rechten Unterbauch*) notiert der Arzt eine bestimmte Anzahl von Punkten. Liegt die Summe der Punkte über einem bestimmten Wert (Schwellenwert), wird eine bestimmte Entscheidung empfohlen (z. B. Betrieb). Für n Symptome $S_1, \dots, S_n$ kann ein Score für Blinddarmentzündung formal beschrieben werden als

$$\text{Diagnose} = \begin{array}{l} \text{Appendizitis, wenn } w_1 S_1 + \dots + w_n S_n > \bar{y}, \text{ negativ} \\ \text{anders.} \end{array}$$

Bei Scores wird also eine lineare Kombination von Symptomwerten mit einem Schwellenwert $\bar{y}$ verglichen. Die Gewichte der Symptome werden mithilfe statistischer Methoden aus Datenbanken extrahiert. Der Vorteil von Scores liegt in ihrer einfachen Anwendung. Die gewichtete Summe der Punkte kann leicht von Hand berechnet werden und für die Diagnose ist kein Computer erforderlich.

Aufgrund der Linearität dieser Methode sind die Ergebnisse zu schwach, um komplexe Zusammenhänge zu modellieren. Da der Beitrag $w_i S_i$ eines Symptoms S_i zum Score unabhängig von den anderen Symptomen berechnet wird, ist klar, dass Scoresysteme keinen „Kontext“ berücksichtigen können. Grundsätzlich können sie nicht zwischen Symptomkombinationen unterscheiden, beispielsweise können sie nicht zwischen der Anzahl der weißen Blutkörperchen eines alten Patienten und der eines jungen Patienten unterscheiden.

Für einen festen gegebenen Satz von Symptomen ist die bedingte Wahrscheinlichkeit viel aussagekräftiger als Scores für die Erstellung von Vorhersagen, da letztere die Abhängigkeiten zwischen verschiedenen Symptomen nicht beschreiben können. Wir können zeigen, dass Scores implizit davon ausgehen, dass alle Symptome unabhängig sind.

Bei der Verwendung von Partituren tritt noch ein weiteres Problem auf. Um eine gute Diagnosequalität zu erreichen, müssen hohe Anforderungen an die Datenbanken gestellt werden, die zur statistischen Ermittlung der Gewichte w_i ~~in der Datenbank~~ ^{in der Datenbank} ~~benötigen~~ ^{benötigen} sie repräsentativ für die Patientengruppe im Einsatzgebiet des Diagnosesystems sein. Dies ist oft nur schwer, wenn nicht sogar unmöglich, zu gewährleisten. In solchen Fällen können Scores und andere statistische Methoden entweder nicht verwendet werden oder weisen eine hohe Fehlerquote auf.

7.3.2 Hybride probabilistische Wissensdatenbank

Komplexe Wahrscheinlichkeitsbeziehungen treten in der Medizin häufig auf. Mit LEXMED lassen sich diese Zusammenhänge gut modellieren und schnell berechnen. Wesentlich ist hierbei der Einsatz probabilistischer Aussagen, mit denen sich unsichere und unvollständige Informationen intuitiv und mathematisch fundiert ausdrücken und verarbeiten lassen. Als typische Abfrage an das Expertensystem kann folgende Frage dienen: „Wie hoch ist die Wahrscheinlichkeit einer Blinddarmentzündung, wenn es sich bei dem Patienten um einen 23-jährigen Mann mit Schmerzen im rechten Unterbauch und einer Anzahl weißer Blutkörperchen von 13.000 handelt?“ Als bedingte Wahrscheinlichkeit formuliert, unter Verwendung der in Tabelle 7.2 auf Seite 133 verwendeten Namen und Wertebereiche für die Symptome, lautet dies:

Tabelle 7.2 Für die Abfrage in LEXMED verwendete Symptome und ihre Werte. Die Anzahl der Werte für Jedes Symptom ist in der mit # gekennzeichneten Spalte aufgeführt.

Symptom	Werte	# Kurz
Geschlecht	<i>Männlich, weiblich</i>	2 <i>Sex2</i>
Alter	<i>0–5, 6–10, 11–15, 16–20, 21–25, 26–35, 36–45, 46–55, 56–65, 65–</i>	10 <i>Alter10</i>
Schmerzen im 1. Quadrizeps.	<i>Ja, nein</i>	2 <i>P1Q2</i>
Schmerzen im 2. Quadrizeps.	<i>Ja, nein</i>	2 <i>P2Q2</i>
Schmerzen im dritten Quadranten.	<i>Ja, nein</i>	2 <i>P3Q2</i>
Schmerzen im vierten Quadranten.	<i>Ja, nein</i>	2 <i>P4Q2</i>
Bewachung	<i>Lokal, global, keine</i>	3 <i>Gua3</i>
Rebound-Zärtlichkeit	<i>Ja, nein</i>	2 <i>Reb2</i>
Schmerzen beim Klopfen	<i>Ja, nein</i>	2 <i>Tippen Sie auf2</i>
Rektaler Schmerz	<i>Ja, nein</i>	2 <i>RecP2</i>
Darmgeräusche	<i>Schwach, normal, erhöht, keine</i>	4 <i>BogenS4</i>
Abnormaler Ultraschall	<i>Ja, nein</i>	2 <i>Sono2</i>
Abnormaler Urinsedim.	<i>Ja, nein</i>	2 <i>Urin2</i>
Temperatur (rektal)	<i>–37,3, 37,4–37,6, 37,7–38,0, 38,1–38,4, 38,5–38,9, 39,0–</i>	6 <i>TRec6</i>
Leukozyten	<i>0–6k, 6k–8k, 8k–10k, 10k–12k, 12k–15k, 15k–20k, 20k–</i>	7 <i>Leuko7</i>
Diagnose	<i>Entzündet, perforiert, negativ, anders</i>	4 <i>Diag4</i>

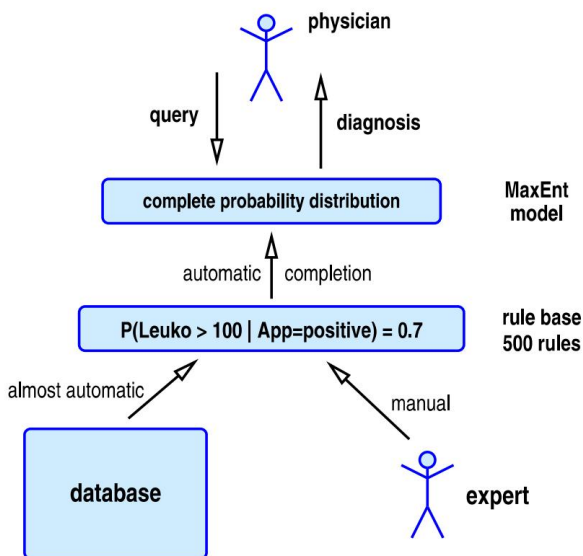
$P(\text{Diag4} = \text{entzündet} \vee \text{Diag4} = \text{perforiert})$

$\text{Geschlecht2} = \text{männlich} \vee \text{Alter } 10 \vee 21\text{--}25 \vee \text{Leuko7} \vee 12.000\text{--}15.000$.

Durch die Verwendung probabilistischer Aussagen ist LEXMED in der Lage, Informationen zu nutzen aus nicht repräsentativen Datenbanken, da diese Informationen ergänzt werden können entsprechend aus anderen Quellen. Zugrunde liegt LEXMED eine Datenbank, die nur enthält Daten über Patienten, deren Blinddarm operativ entfernt wurde. Mit statistischen Methoden werden (ca. 400) Regeln generiert, die das in der Datenbank enthaltene Wissen in einer abstrahierten Form zusammenfassen[ES99]. Weil es keine Patienten gibt in dieser Datenbank, bei denen der Verdacht auf eine Blinddarmentzündung bestand, die jedoch negative Diagnosen hatten (d. h. keine Behandlung benötigten),⁶ gibt es keine Erkenntnisse über negative Patienten in der Datenbank. Daher muss Wissen aus anderen Quellen hinzugefügt werden. In LEXMED werden daher die aus der Datenbank erfassten Regeln durch ergänzt (ca. 100) Regeln von medizinischen Experten und der medizinischen Literatur. Das führt zu einer hybride probabilistische Datenbank, die aus Daten extrahiertes Wissen enthält sowie von Experten explizit formuliertes Wissen. Weil es beide Arten von Regeln gibt

⁶Diese negativen Diagnosen werden als „unspezifische Bauchschmerzen“ (NSAP) bezeichnet.

Abb. 7.5 Aus Daten und Expertenwissen werden probabilistische Regeln generiert, die in eine Regelbasis (Wissensbasis) integriert und schließlich mit der MaxEnt-Methode vervollständigt werden



Als bedingte Wahrscheinlichkeiten formuliert (siehe zum Beispiel (7.14) auf Seite 138), können sie leicht kombiniert werden, wie in Abb. 7.5 und mit weiteren Einzelheiten in Abb. 7.7 auf Seite 136 dargestellt.

LECMED berechnet die Wahrscheinlichkeiten verschiedener Diagnosen anhand der Wahrscheinlichkeitsverteilung aller relevanten Variablen (siehe Tabelle 7.2 auf Seite 133). Da alle 14 in LEXMED verwendeten Symptome und die Diagnosen als diskrete Variablen modelliert werden (sogar kontinuierliche Variablen wie der Leukozytenwert werden in Bereiche unterteilt), kann die Größe der Verteilung (d. h. die Größe des Ereignisraums) anhand der Tabelle bestimmt werden 7.2 auf Seite 133 als Produkt der Anzahl der Werte aller Symptome,

$$10 \cdot 2 \cdot 10 \cdot 3 \cdot 4 \cdot 6 \cdot 7 \cdot 4 = 20\,643\,840$$

Elemente. Aufgrund der Normalisierungsbedingung aus Satz 7.1 hat er somit 20.643.839 unabhängige Werte. Jeder Regelsatz mit weniger als 20.643.839 Wahrscheinlichkeitswerten beschreibt diesen Ereignisraum potenziell nicht vollständig. Um jede beliebige Frage beantworten zu können, benötigt das Expertensystem eine vollständige Verteilung. Die Konstruktion einer so umfangreichen, konsistenten Verteilung mit statistischen Methoden ist sehr schwierig.⁷ Von einem menschlichen Experten alle 20.643.839 Werte für die Verteilung (anstelle der oben genannten 100 Regeln) zu verlangen, wäre praktisch unmöglich.

Hier kommt die MaxEnt-Methode ins Spiel. Die Verallgemeinerung von etwa 500 Regeln zu einem vollständigen Wahrscheinlichkeitsmodell erfolgt in LEXMED durch Maximierung der Entropie mit den 500 Regeln als Einschränkungen. Eine effiziente Kodierung der resultierenden MaxEnt-Verteilung führt zu Antwortzeiten für die Diagnose von etwa einer Sekunde.

⁷Die Aufgabe, aus einem Datensatz eine Funktion zu generieren, wird als maschinelles Lernen bezeichnet. Wir werden dies in Kap. ausführlich behandeln. 8.

Personal Details	unknown	values	
Gender	☒	☐ male ☐ female	?
Age-group	☐	☐ 0-5 ☐ 6-10 ☐ 11-15 ☐ 16-20 ☒ 21-25 ☐ 26-35 ☐ 36-45 ☐ 46-55 ☐ 56-65 ☐ 65-	?
Results of examination	not done	values	
1st quadrant	☒	☐ yes ☐ no	?
2nd quadrant	☒	☐ yes ☐ no	?
3rd quadrant	☐	☒ yes ☐ no	?
4th quadrant	☒	☐ yes ☐ no	?
guarding	☐	☒ local ☐ global ☐ none	?
rebound tenderness	☐	☒ yes ☐ no	?
pain on tapping	☐	☒ yes ☐ no	?
rectal pain	☒	☐ yes ☐ no	?
bowel sounds	☐	☐ weak ☒ normal ☐ increased ☐ none	?
abnormal ultrasound	☒	☐ yes ☐ no	?
abnormal urine sediment	☒	☐ yes ☐ no	?
temperature range (rectal)	☐	☐ -37.3 ☐ 37.4-37.6 ☐ 37.7-38.0 ☐ 38.1-38.4 ☒ 38.5-38.9 ☐ 39.0-	?
leucocyte count	☐	☐ 0-6k ☐ 6k-8k ☐ 8k-10k ☐ 10k-12k ☒ 12k-15k ☐ 15k-20k ☐ 20k-	?

Result of the PIT diagnosis				
Diagnosis	App. inflamed	App. perforated	Negative	Other
Probability	0.70	0.17	0.06	0.07

Abb. 7.6 Die LEXMED- Eingabemaske zur Eingabe der untersuchten Symptome und darunter die Ausgabe der resultierenden Diagnosewahrscheinlichkeiten

7.3.3 Anwendung von LEXMED

Die Nutzung von LEXMED ist einfach und selbsterklärend. Der Arzt besucht für eine Homepage von LEXMED unter ⁸ automatische Diagnose den Arzt www.lexmed.de. gibt die Ergebnisse seiner Untersuchung in das Eingabeformular in Abb. 7.6 ein. Nach ein bis zwei Sekunden erhält er die Wahrscheinlichkeiten für die vier verschiedenen Diagnosen sowie einen Behandlungsvorschlag (Abschn. 7.3.5). Fehlen bestimmte Untersuchungsergebnisse als Eingabe (zum Beispiel die Sonogramm-Ergebnisse), wählt der Arzt den Eintrag *nicht untersucht*. Natürlich ist die Sicherheit der Diagnose umso höher, je mehr Symptomwerte eingegeben werden.

⁸Eine Version mit eingeschränkter Funktionalität ist ohne Passwort zugänglich.

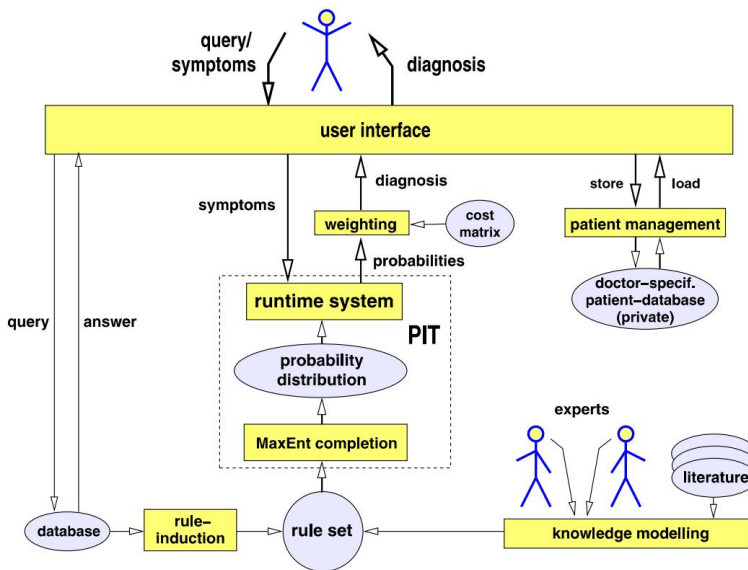


Abb. 7.7 Regeln werden sowohl aus der Datenbank als auch aus Expertenwissen generiert. Daraus erstellt MaxEnt eine vollständige Wahrscheinlichkeitsverteilung. Für eine Benutzerabfrage wird die Wahrscheinlichkeit jeder möglichen Diagnose berechnet. Anhand der Kostenmatrix (siehe Abschn. 7.3.5) wird dann eine Entscheidung vorgeschlagen.

Jeder registrierte Benutzer hat Zugriff auf eine private Patientendatenbank, in der Eingabedaten archiviert werden können. Somit können Daten und Diagnosen früherer Patienten problemlos mit denen eines neuen Patienten verglichen werden.

7.3.4 Funktion von LEXMED

Wissen wird mithilfe probabilistischer Aussagen formalisiert. Zum Beispiel der Vorschlag

$$P(\text{Leuko7} > 20000 | \text{Diag4} = \text{entzündet}) = 0,09$$

gibt eine Häufigkeit von 9 % für einen Leukozytenwert von mehr als 20.000 bei einer Blinddarmentzündung an.⁹

Lernen von Regeln durch statistische Induktion Die

Rohdaten in der Datenbank von LEXMED enthalten 54 verschiedene (anonymisierte) Werte für 14.646 Patienten. Wie bereits erwähnt, sind in dieser Datenbank nur Patienten enthalten, deren Blinddarm operativ entfernt wurde. Von den 54 in der Datenbank verwendeten Attributen sind nach einer statistischen Analyse die 14 Symptome in Tabelle 7.2 auf Seite 133 aufgeführt

⁹ Anstelle einzelner Zahlenwerte können hier auch Intervalle verwendet werden (z. B. [0,06, 0,12]).

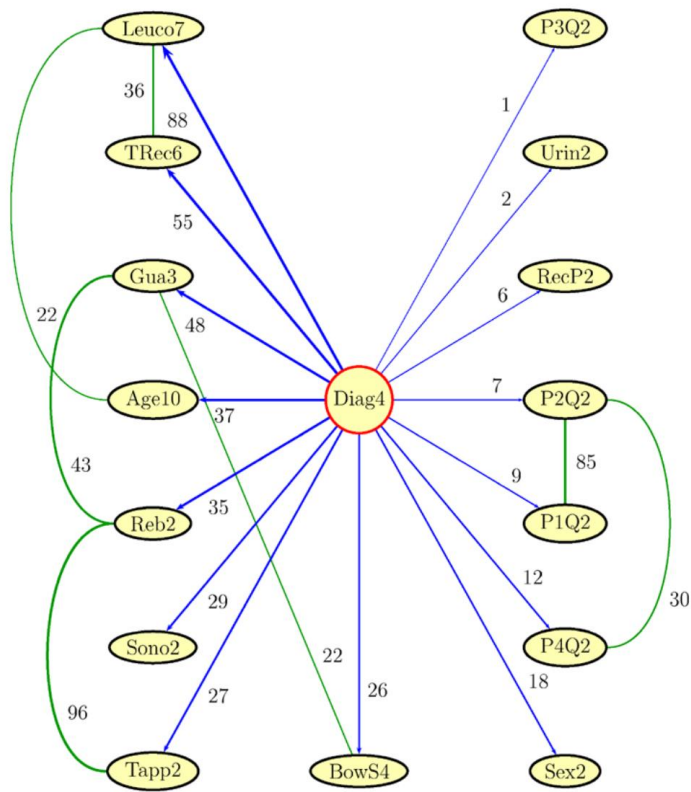


Abb. 7.8 Aus der Datenbank berechneter Abhängigkeitsgraph

wurden benutzt. Aus dieser Datenbank werden nun in zwei Schritten die Regeln erstellt. Im ersten Schritt wird die Abhängigkeitsstruktur der Symptome ermittelt. Im zweiten Schritt wird diese Struktur mit den entsprechenden Wahrscheinlichkeitsregeln gefüllt.¹⁰

Bestimmung des Abhängigkeitsgraphen Der Graph in Abb. 7.8 enthält für jede Variable (das Symptom und die Diagnose) einen Knoten und gerichtete Kanten, die verschiedene Knoten verbinden. Die Dicke der Kanten zwischen den Variablen stellt ein Maß für die statistische Abhängigkeit bzw. Korrelation der Variablen dar. Die Korrelation zweier unabhängiger Variablen ist gleich Null. Die Paarkorrelation für jedes der 14 Symptome mit *Diag4* wurde berechnet und in der Grafik aufgeführt. Darüber hinaus wurden alle dreifachen Korrelationen zwischen der Diagnose und zwei Symptomen berechnet. Von diesen wurden nur die stärksten Werte als zusätzliche Kanten zwischen den beiden beteiligten Symptomen eingezeichnet.

¹⁰Für eine systematische Einführung in maschinelles Lernen verweisen wir den Leser auf Kap. 8.

```

1 P([Leuco7=0-6k] | [Diag4=negativ] * [Alter10=16-20]) = [0,132,0,156];
2 P([Leuco7=6-8k] | [Diag4=negativ] * [Alter10=16-20]) = [0,257,0,281];
3 P([Leuco7=8-10k] | [Diag4=negativ] * [Alter10=16-20]) = [0,250,0,274];
4 P([Leuco7=10-12k] | [Diag4=negativ] * [Alter10=16-20]) = [0,159,0,183];
5 P([Leuco7=12-15k] | [Diag4=negativ] * [Alter10=16-20]) = [0,087,0,112];
6 P([Leuco7=15-20k] | [Diag4=negativ] * [Alter10=16-20]) = [0,032,0,056];
7 P([Leuco7=20k-] | [Diag4=negativ] * [Alter10=16-20]) = [0,000,0,023];
8 P([Leuco7=0-6k] | [Diag4=negativ] * [Alter10=21-25]) = [0,132,0,172];
9 P([Leuco7=6-8k] | [Diag4=negativ] * [Alter10=21-25]) = [0,227,0,266];
10 P([Leuco7=8-10k] | [Diag4=negativ] * [Alter10=21-25]) = [0,211,0,250];
11 P([Leuco7=10-12k] | [Diag4=negativ] * [Alter10=21-25]) = [0,166,0,205];
12 P([Leuco7=12-15k] | [Diag4=negativ] * [Alter10=21-25]) = [0,081,0,120];
13 P([Leuco7=15-20k] | [Diag4=negativ] * [Alter10=21-25]) = [0,041,0,081];
14 P([Leuco7=20k-] | [Diag4=negativ] * [Alter10=21-25]) = [0,004,0,043];

```

Abb. 7.9 Einige der LEXMED- Regeln mit Wahrscheinlichkeitsintervallen. „*“ steht hier für „ $\dot{\vee}$ “.

Abschätzen der Regelwahrscheinlichkeiten Die Struktur des Abhängigkeitsgraphen beschreibt die Struktur der erlernten Regeln.¹¹ Die Regeln sind dabei unterschiedlich komplex: Es gibt Regeln, die nur die Verteilung der möglichen Diagnosen beschreiben (A-priori-Regeln, zum Beispiel (7.13)), Regeln, die die Abhängigkeit zwischen beschreiben die Diagnose und ein Symptom (Regeln mit einfachen Bedingungen, zum Beispiel (7.14)), und schließlich Regeln, die die Abhängigkeit zwischen der Diagnose und zwei Symptomen beschreiben, wie in Abb. 7.9 in PIT- Syntax dargestellt.

$$P(\text{Diag4} = \text{entzündet}) = 0,40, P \quad (7.13)$$

$$(\text{Sono2} = \text{ja} | \text{Diag4} = \text{entzündet}) = 0,43, \quad (7.14)$$

$$P(P4Q2 = \text{ja} | \text{Diag4} = \text{entzündet} \dot{\vee} P2Q2 = \text{ja}) = 0,61. \quad (7.15)$$

Um die Kontextabhängigkeit des gespeicherten Wissens möglichst gering zu halten, Alle Regeln enthalten die Diagnose in ihren Bedingungen und nicht als Schlussfolgerungen. Das ist ganz ähnlich dem Aufbau vieler medizinischer Bücher mit Formulierungen des Art „Bei einer Blinddarmentzündung sehen wir normalerweise . . . “. Wie zuvor in Beispiel 7.6 gezeigt Auf Seite 121 stellt dies jedoch kein Problem dar, da die Bayesian-Methode verwendet wird Mithilfe der Formel bringt LECMED diese Regeln automatisch in die richtige Form.

Die numerischen Werte für diese Regeln werden durch Zählen ihrer Häufigkeit geschätzt die Datenbank. Beispielsweise ergibt sich der Wert in (7.14) durch Zählen und Rechnen

$$\frac{|\text{Diag4} = \text{entzündet} \dot{\vee} \text{Sono2} = \text{ja}|}{|\text{Diag4} = \text{entzündet}|} = 0,43.$$

¹¹Der Unterschied zu einem Bayes'schen Netzwerk besteht beispielsweise darin, dass die Regeln ausgestattet sind mit Wahrscheinlichkeitsintervallen und dass erst nach Anwendung des Prinzips der maximalen Entropie ein Unikat vorliegt Wahrscheinlichkeitsmodell erstellt.

Expertenregeln

Denn die Appendizitis-Datenbank enthält nur Patienten, die sich einer Blinddarmentzündung unterzogen haben Operation, Regeln für unspezifische Bauchschmerzen (*NSAP*) erhalten ihre Werte von Vorschläge medizinischer Experten. Die Erfahrungen in LEXMED bestätigen, dass die Wahrscheinlichkeitsregeln leicht zu lesen sind und direkt in natürliche Sprache übersetzt werden können.

Aussagen medizinischer Experten zu Häufigkeitszusammenhängen bestimmter Symptome und die Diagnose, sei es aus der Literatur oder als Ergebnis eines Interviews, kann können daher mit geringem Aufwand in die Regelbasis integriert werden. Zur Modellierung der Unsicherheit von Expertenwissen hat sich die Verwendung von Wahrscheinlichkeitsintervallen bewährt.

Das Expertenwissen wurde in erster Linie von den beteiligten Chirurgen Dr.

Rampf und Dr. Hontschik und ihre Veröffentlichungen [Hon94].

Sobald die Expertenregeln erstellt wurden, ist die Regelbasis fertig. Dann wird das vollständige Wahrscheinlichkeitsmodell mit der Methode der maximalen Entropie berechnet PIT-System.

Diagnoseabfragen

Mithilfe seines effizient hinterlegten Wahrscheinlichkeitsmodells berechnet LEXMED die Wahrscheinlichkeiten für die vier möglichen Diagnosen innerhalb weniger Sekunden. Wir gehen zum Beispiel davon aus, dass folgende Ausgabe:

Diagnose	Ergebnisse der PIT-Diagnose		
	Blinddarm entzündet	Blinddarm perforiert	0,24 0,16
	Negativ	Andere	
Wahrscheinlichkeit	0,57	0,03	

Auf der Grundlage dieser vier Wahrscheinlichkeitswerte muss eine Entscheidung getroffen werden, einen dieser Werte zu verfolgen Die vier Behandlungen: Operation, Notoperation, stationäre Beobachtung oder ambulante Beobachtung.¹²

Während die Wahrscheinlichkeit für eine negative Diagnose in diesem Fall hoch ist

Da die Entscheidung überwiegt, ist es keine gute Entscheidung, den Patienten als gesund nach Hause zu schicken.

Wir können das deutlich sehen, auch wenn die Wahrscheinlichkeiten der Diagnosen hoch waren berechnet, die Diagnose ist noch nicht abgeschlossen.

Die Aufgabe besteht nun vielmehr darin, aus diesen Wahrscheinlichkeiten eine optimale Entscheidung abzuleiten. Zu Zu diesem Zweck kann der Nutzer von LEXMED eine empfohlene Entscheidung berechnen lassen.

7.3.5 Risikomanagement mithilfe der Kostenmatrix

Wie können nun die berechneten Wahrscheinlichkeiten optimal in Entscheidungen umgesetzt werden?

Ein naiver Algorithmus würde jeder Diagnose eine Entscheidung zuweisen und letztendlich auswählen

die Entscheidung, die der höchsten Wahrscheinlichkeit entspricht. Gehen Sie davon aus, dass die berechnete

Die Wahrscheinlichkeiten für die Diagnose *Blinddarmentzündung* (entzündet oder perforiert) liegen bei 0,40 und bei 0,55 für die Diagnose „negativ“ und 0,05 für die Diagnose „andere“. Ein naiver Algorithmus würde es tun

Wählen Sie nun die (zu riskante) Entscheidung „keine Operation“, da diese der Diagnose mit der höheren Wahrscheinlichkeit entspricht. Eine bessere Methode besteht darin, die Kosten zu vergleichen

¹²Ambulante Beobachtung bedeutet, dass der Patient zu Hause entlassen wird.

Tabelle 7.3 Die Kostenmatrix von LEXMED zusammen mit den berechneten Diagnosewahrscheinlichkeiten eines Patienten

Therapie	Wahrscheinlichkeit verschiedener Diagnosen				
	<i>Entzündet</i>	<i>Perforiert</i>	<i>Negativ</i>	<i>Andere</i>	
	0,25	0,15	0,55	0,05	
Betrieb	0	500	5800	6000	3565
Notoperation	500	0	6300	6500	3915
Ambulante Beobachtung.	12000	150000	0	16500	26325
Andere	3000	5000	1300	0	2215
Stationärer Beobachter	3500	7000	400	600	2175

der möglichen Fehler, die bei jeder Entscheidung auftreten können. Der Fehler wird in der quantifiziert Form von „(hypothetischen) zusätzlichen Kosten der aktuellen Entscheidung im Vergleich zum Optimum“. Die angegebenen Werte beinhalten die Kosten für das Krankenhaus, für die Versicherung, dem Patienten (z. B. Risiko postoperativer Komplikationen) und anderen Parteien (z. B. Abwesenheit vom Arbeitsplatz) unter Berücksichtigung langfristiger Folgen.

Diese Kosten sind in Tabelle 7.3 aufgeführt.

Die Eingaben werden abschließend für jede Entscheidung gemittelt, also beim Treffen summiert Berücksichtigung ihrer Frequenzen. Diese sind in der letzten Spalte der Tabelle 7.3 aufgeführt. Abschließend wird die Entscheidung mit den geringsten durchschnittlichen Fehlerkosten vorgeschlagen. In Tabelle 7.3 Die Matrix wird zusammen mit dem für einen Patienten berechneten Wahrscheinlichkeitsvektor angegeben (in diesem Fall: (0,25, 0,15, 0,55, 0,05)). Die letzte Spalte der Tabelle enthält das Ergebnis von die Berechnungen der durchschnittlich erwarteten Kosten der Fehler. Der Wert der *Operation* in der ersten Zeile errechnet sich somit zu $0,25 \cdot 0 + 0,15 \cdot 500 + 0,55 \cdot 5800 + 0,05 \cdot 6000 = 3565$, ein gewichteter Durchschnitt aller Kosten. Die optimalen Entscheidungen werden mit (Zusatz-)Kosten von 0 eingetragen. Das System entscheidet sich für die Behandlung mit dem minimalen Durchschnittskosten. Es handelt sich somit um ein Beispiel für einen kostenorientierten Agenten.

Kostenmatrix im binären Fall

Um die Kostenmatrix und das Risikomanagement besser zu verstehen, werden wir nun die einschränken LEXMED- System zur zweiwertigen Entscheidung zwischen der Diagnose *Blinddarmentzündung* und *NSAP*. Als mögliche Behandlungen sind nur *Operation* mit Wahrscheinlichkeit p_1 und *ambulante Behandlung* möglich. Es kann *eine Beobachtung* mit der Wahrscheinlichkeit p_2 gewählt werden. Die Kostenmatrix ist somit eine 2×2 Matrix des Formulars

$$\begin{matrix} & 0 & k_2 \\ k_1 & 0 & \end{matrix}$$

Die beiden Nullen in der Diagonale stehen für die jeweils richtige *Entscheidungsoperation* der *Blinddarmentzündung* und *ambulante Beobachtung* auf *NSAP*. Der Parameter k_2 steht für die zu erwartenden Kosten, die entstehen, wenn ein Patient keinen entzündeten Blinddarm hat operiert. Dieser Fehler wird als falsch positiv bezeichnet. Andererseits die Entscheidung

Eine ambulante Beobachtung ist bei einer Blinddarmentzündung falsch negativ. Der Wahrscheinlichkeitsvektor $(p_1, p_2)^T$ dieser Matrix multipliziert und wir erhalten den Vektor

$$(2 \text{ re. } 2 \text{ p.}, 1 \text{ re. } p. \text{ } 1)^T$$

mit den durchschnittlichen Mehrkosten für die beiden möglichen Behandlungen. Da die Entscheidung nur das Verhältnis der beiden Komponenten berücksichtigt, kann der Vektor mit einem beliebigen Skalarfaktor multipliziert werden. Wir wählen $1/k_1$ und erhalten $((k_2/k_1)p_2, p_1)$. Daher ist hier nur die Beziehung $k = k_2/k_1$ relevant. Das gleiche Ergebnis erhält man mit der einfacheren Kostenmatrix

$$\begin{pmatrix} 0 & k & 1 \\ 0 & & \end{pmatrix},$$

die nur die Variable k enthält. Dieser Parameter ist sehr wichtig, da er das Risikomanagement bestimmt. Durch die Änderung von k können wir den „Arbeitspunkt“ des Diagnosesystems anpassen. Für $k \rightarrow \infty$ befindet sich das System in einer äußerst riskanten Umgebung, da kein Patient jemals operiert wird, mit der Konsequenz, dass es keine falsch-positiven Klassifizierungen, aber viele falsch-negative Klassifizierungen liefert. Bei $k = 0$ sind die Verhältnisse genau umgekehrt und alle Patienten werden operiert.

7.3.6 Leistung

LEUXMED ist für den Einsatz in einer Arztpraxis oder einem Krankenwagen bestimmt.

Voraussetzung für die Anwendung von LEXMED sind akute Bauchschmerzen über mehrere Stunden (jedoch weniger als fünf Tage). Darüber hinaus ist LEXMED (derzeit) auf Blinddarmentzündung spezialisiert, was bedeutet, dass das System für andere Krankheiten nur sehr wenige Informationen benötigt.

Im Rahmen einer prospektiven Studie wurde eine repräsentative Datenbank mit 185 Fällen im 14. Nothelfer Krankenhaus erstellt. Es enthält die Patienten des Krankenhauses, die nach mehreren Stunden akuter Bauchschmerzen und Verdacht auf Blinddarmentzündung in die Klinik kamen. Von diesen Patienten werden die Symptome und die Diagnose (im Falle einer Operation anhand einer Gewebeprobe überprüft) notiert.

Konnten die Patienten nach mehrstündigem oder 1–2-tägigem Aufenthalt beschwerdefrei nach Hause (ohne Operation) entlassen werden, wurde anschließend telefonisch erfragt, ob der Patient beschwerdefrei blieb oder ob eine positive Diagnose vorliegt anschließende Behandlung.

Um die Darstellung zu vereinfachen und einen besseren Vergleich mit ähnlichen Studien zu ermöglichen, wurde LEXMED auf die zweiwertige Unterscheidung zwischen *Appendizitis* und *NSAP beschränkt*, wie in Abschn. 7.3.5. Jetzt wird k zwischen Null und Unendlich variiert und für jeden Wert von k werden die *Sensitivität* und *Spezifität* anhand der Testdaten gemessen. Empfindlichkeitsmaße

$$P(\text{positiv klassifiziert} | \text{positiv}) = \frac{|\text{positiv und als positiv eingestuft}|}{|\text{positiv}|},$$

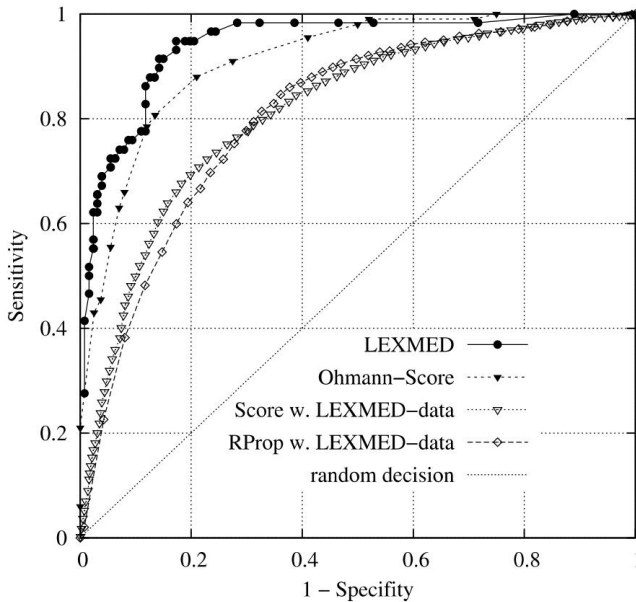


Abb. 7.10 ROC-Kurve von LEXMED im Vergleich zum Ohmann-Score und zwei weiteren Modellen

das heißt, der relative Anteil der positiven Fälle, die korrekt identifiziert wurden. Sie gibt an, wie empfindlich das Diagnosesystem ist. Spezifität hingegen misst

$$P(\text{klassifiziert negativ}|\text{negativ}) = \frac{|\text{negativ und als negativ eingestuft}|}{|\text{negativ}|},$$

das heißt, der relative Anteil der korrekt identifizierten Negativfälle.

Die Ergebnisse der Sensitivität und Spezifität geben wir in Abb. 7.10 für $0 \leq k < \infty$. Diese Kurve wird als *ROC-Kurve* oder Receiver Operating Characteristic bezeichnet.

Bevor wir zur Analyse der Qualität von LEXMED kommen, noch ein paar Worte zur

Bedeutung der ROC-Kurve. Zur Orientierung ist die Linie eingezeichnet, die das Diagramm diagonal

hinauf verläuft. Alle Punkte auf dieser Linie entsprechen einer Zufallsentscheidung. Beispielsweise

entspricht der Punkt (0,2, 0,2) einer Spezifität von 0,8 bei einer Sensitivität von 0,2. Wir

können dies ganz einfach erreichen, indem wir einen neuen Fall, ohne ihn anzusehen,

mit Wahrscheinlichkeiten von 0,2 für positiv und 0,8 für negativ klassifizieren. Jedes

wissensbasierte Diagnosesystem muss daher einen ROC generieren, der deutlich über der Diagonale

liegt. Interessant sind auch die Extremwerte in der ROC-Kurve. Im Punkt (0, 0) schneiden

sich alle drei Kurven. Das entsprechende Diagnosesystem würde alle Fälle als negativ

klassifizieren. Der andere Extremwert (1, 1) entspricht einem System, das entscheiden

würde, die Operation bei jedem Patienten durchzuführen und hat daher eine Sensitivität

von 1. Wir könnten die ROC-Kurve als charakteristische Kurve für zweiwertige

Diagnosesysteme bezeichnen. Das ideale Diagnosesystem hätte eine Kennlinie, die nur

aus dem Punkt (0, 1) besteht und somit 100 % Spezifität und 100 % Sensitivität aufweist.

Lassen Sie uns nun die ROC-Kurve analysieren. Bei einer Sensitivität von 88 % erreicht LEXMED eine Spezifität von 87 % ($k = 0,6$). Zum Vergleich wird der Ohmann-Score, ein etablierter, bekannter Score für Appendizitis, angegeben [OMYL96, ZSR+99]. Da LEXMED fast überall über oder links vom Ohmann-Score liegt, ist die durchschnittliche Diagnosequalität deutlich besser. Dies ist nicht überraschend, da die Ergebnisse einfach zu schwach sind, um interessante Aussagen zu modellieren. In Abschn. 8.6 und in Übung 8.16 auf Seite 219 werden wir zeigen, dass Scores äquivalent zum Spezialfall des naiven Bayes sind, also der Annahme, dass alle Symptome paarweise unabhängig sind, wenn die Diagnose bekannt ist. Beim Vergleich von LEXMED mit Scores ist jedoch zu erwähnen, dass für den Ohmann-Score eine statistisch repräsentative Datenbank verwendet wurde, für LEXMED jedoch eine nicht repräsentative, mit Expertenwissen angereicherte Datenbank. Um einen Eindruck von der Qualität der LEXMED- Daten im Vergleich zu den Ohmann-Daten zu bekommen, wurde ein linearer Score nach der Methode der kleinsten Quadrate berechnet (siehe Abschn. 9.4.1), der auch zum Vergleich herangezogen wird. Darüber hinaus wurde ein neuronales Netzwerk auf den LEXMED- Daten mit dem RProp-Algorithmus trainiert (siehe Abschn. 9.5). Die Stärke der Kombination von Daten und Expertenwissen zeigt sich deutlich im Unterschied zwischen der LEXMED- Kurve und den Kurven des Score-Systems und

7.3.7 Anwendungsbereiche und Erfahrungen

LEXMED sollte nicht das Urteil eines erfahrenen Chirurgen ersetzen. Da im klinischen Umfeld jedoch nicht immer ein Facharzt zur Verfügung steht, bietet eine LEXMED- Anfrage eine fundierte Zweitmeinung. Besonders interessant und lohnenswert ist der Einsatz des Systems im klinischen Rettungswagen und bei niedergelassenen Ärzten.

Die Lernfähigkeit von LEXMED, die die Berücksichtigung weiterer Symptome, weiterer Patientendaten und weiterer Regeln ermöglicht, eröffnet auch in der Klinik neue Möglichkeiten. Für besonders seltene und schwierig zu diagnostizierende Gruppen, beispielsweise Kinder unter sechs Jahren, kann LEXMED Daten von Kinderärzten oder anderen speziellen Datenbanken nutzen, um auch erfahrene Chirurgen zu unterstützen.

Neben dem direkten Einsatz in der Diagnostik unterstützt LEXMED auch Maßnahmen zur Qualitätssicherung. Beispielsweise können Versicherungen die Diagnosequalität von Krankenhäusern mit der von Expertensystemen vergleichen. Durch die Weiterentwicklung der in LEXMED erstellten Kostenmatrix (mit Zustimmung von Ärzten, Versicherungen und Patienten) wird die Qualität von Arztdiagnosen, Computerdiagnosen und anderen medizinischen Einrichtungen besser vergleichbar.

LEXMED hat einen neuen Weg zum Aufbau automatischer Diagnosesysteme aufgezeigt. Mithilfe der Sprache der Wahrscheinlichkeitstheorie und des MaxEnt-Algorithmus wird induktiv statistisch abgeleitetes Wissen mit Wissen von Experten und aus der Literatur kombiniert. Der auf probabilistischen Modellen basierende Ansatz ist theoretisch elegant, allgemein anwendbar und hat in einer kleinen Studie zu sehr guten Ergebnissen geführt.

Seit 1999 ist LEXMED im Nothelfer-Krankenhaus 14 in Weingarten im praktischen Einsatz und hat sich dort sehr gut bewährt. Es ist auch unter www.lexmed.de erhältlich, natürlich ohne Gewähr. Die Diagnosequalität ist vergleichbar mit der eines erfahrenen Chirurgen und damit besser als die eines durchschnittlichen Allgemeinmediziners oder eines unerfahrenen Arztes in der Klinik.

Trotz dieses Erfolgs zeigt sich, dass es sehr schwierig ist, ein solches System im deutschen Medizinsystem kommerziell zu vermarkten. Ein Grund dafür ist, dass es keinen freien Markt gibt, der durch seine Selektionsmechanismen eine bessere Qualität (hier bessere Diagnosen) fördert. Darüber hinaus ist in der Medizin auch im Jahr 2010 noch nicht die Zeit für einen breiten Einsatz intelligenter Techniken gekommen. Eine Ursache hierfür könnten konservative Lehren in dieser Hinsicht an deutschen medizinischen Fakultäten sein.

Ein weiteres Thema ist der Wunsch vieler Patienten nach persönlicher Beratung und Betreuung durch den Arzt sowie die Befürchtung, dass der Patient mit der Einführung von Expertensystemen nur noch mit der Maschine kommuniziert. Diese Angst ist jedoch völlig unbegründet. Auch auf lange Sicht können medizinische Expertensysteme den Arzt nicht ersetzen. Sie können jedoch ebenso wie Laserchirurgie und Magnetresonanztomographie für alle Beteiligten vorteilhaft eingesetzt werden. Seit dem ersten medizinischen Computer-Diagnosesystem von de Dombal im Jahr 1972 sind fast 40 Jahre vergangen. Es bleibt abzuwarten, ob die Medizin noch weitere 40 Jahre warten wird, bis die Computerdiagnostik ein etabliertes medizinisches Instrument wird.

7.4 Argumentation mit Bayes'schen Netzwerken

Auf ein Problem des Wahrscheinlichkeitsschlusses in der Praxis wurde bereits in Abschn. 7.1. Werden d Variablen $X_1, \dots, X_d$ mit jeweils n Werten verwendet, dann die zugehörigen d Die Wahrscheinlichkeitsverteilung-Gesamtwerte. Dies bedeutet, dass im schlimmsten Fall die benötigt n Speicher und die Rechenzeit zur Bestimmung der angegebenen Wahrscheinlichkeiten wächst exponentiell mit der Anzahl der Variablen.

In der Praxis sind die Anwendungen meist sehr strukturiert und die Verteilung weist viele Redundanzen auf. Dies bedeutet, dass er mit geeigneten Methoden deutlich reduziert werden kann. Der Einsatz von Bayes'schen Netzwerken hat sich hier bewährt und gehört zu den KI-Techniken, die in der Praxis erfolgreich eingesetzt werden. Bayesianische Netze nutzen das Wissen über die Unabhängigkeit von Variablen, um das Modell zu vereinfachen.

7.4.1 Unabhängige Variablen

Im einfachsten Fall sind alle Variablen paarweise unabhängig und das ist auch der Fall

$$P(X_1, \dots, X_d) = P(X_1) \cdot P(X_2) \cdots P(X_d).$$

Alle Einträge in der Verteilung können somit aus den d -Werten $P(X_1), \dots, P(X_d)$ berechnet werden. Interessante Anwendungen können jedoch meist nicht modelliert werden, weil bedingte Wahrscheinlichkeiten trivial werden.¹³ Wegen

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

¹³Bei der naiven Bayes-Methode wird die Unabhängigkeit aller Attribute vorausgesetzt und diese Methode wurde erfolgreich auf die Textklassifizierung angewendet (siehe Abschn. 8.6).

Alle bedingten Wahrscheinlichkeiten werden auf die A-priori-Wahrscheinlichkeiten reduziert. Interessanter wird die Situation, wenn nur ein Teil der Variablen unter bestimmten Bedingungen unabhängig oder abhängig ist. Für die Argumentation in der KI sind die Abhängigkeiten zwischen Variablen wichtig und müssen genutzt werden.

Wir möchten die Argumentation mit Bayes'schen Netzwerken anhand eines einfachen und sehr anschaulichen Beispiels von J. Pearl [Pea88] skizzieren, das durch [RN10] bekannt wurde und mittlerweile zum KI-Grundwissen gehört.

Beispiel 7.7 (Alarm-Beispiel) Bob, der Single ist, hat in seinem Haus eine Alarmanlage zum Schutz vor Einbrechern installieren lassen. Bob kann den Alarm nicht hören, wenn er im Büro arbeitet. Deshalb hat er seine beiden Nachbarn, John im Haus nebenan links und Mary im Haus rechts, gebeten, ihn in seinem Büro anzurufen, wenn sie seinen Alarm hören. Nach ein paar Jahren weiß Bob, wie zuverlässig John und Mary sind und modelliert ihr Anrufverhalten mithilfe der bedingten Wahrscheinlichkeit wie folgt.¹⁴

$$P(J|A) = 0,90 \quad P(M|A) = 0,70, \quad P(J|\neg A) = 0,05 \quad P(M|\neg A) = 0,01.$$

Da Mary schwerhörig ist, hört sie den Alarm häufiger als John.

Allerdings verwechselt John manchmal die Alarmanlage in Bobs Haus mit der Alarmanlage in anderen Häusern. Der Alarm wird durch einen Einbruch ausgelöst, kann aber auch durch ein (schwaches) Erdbeben ausgelöst werden, was zu einem Fehlalarm führen kann, da Bob nur in seinem Büro über Einbrüche Bescheid wissen möchte. Diese Beziehungen werden modelliert von

$$P(A|Bur, Ohr) = 0,95,$$

$$P(A|Bur, \neg Ear) = 0,94,$$

$$P(A|\neg Bur, Ohr) = 0,29,$$

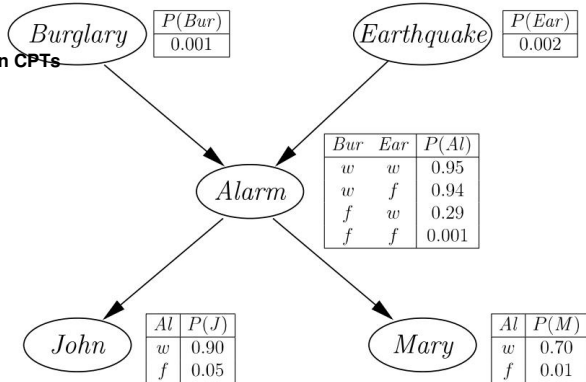
$$P(A|\neg Bur, \neg Ear) = 0,001,$$

sowie die A-priori-Wahrscheinlichkeiten $P(Bur) = 0,001$ und $P(Ear) = 0,002$. Diese beiden Variablen sind unabhängig voneinander, da Erdbebenpläne nicht auf der Grundlage der Gewohnheiten von Einbrechern erstellt werden und es umgekehrt keine Möglichkeit gibt, Erdbeben vorherzusagen, sodass Einbrecher nicht die Möglichkeit haben, ihren Zeitplan entsprechend festzulegen.

Nun werden Abfragen gegen diese Wissensdatenbank durchgeführt. Bob könnte beispielsweise an $P(Bur|J \vee M)$, $P(J|Bur)$ oder $P(M|Bur)$ interessiert sein. Das heißt, er möchte wissen, wie empfindlich die Variablen J und M auf eine Einbruchsmeldung reagieren.

¹⁴Die binären Variablen J und M stehen für die beiden Ereignisse „John ruft“ bzw. „Maria ruft“, A für „Alarmsirene ertönt“, Bur für „Einbruch“ und Ohr für „Erdbeben“.

Abb. 7.11 Bayesianisches
Netzwerk für das
Alarmbeispiel mit den zugehörigen CPTs



7.4.2 Grafische Darstellung von Wissen als Bayesianisches Netzwerk

Wir können die praktische Arbeit erheblich vereinfachen, indem wir Wissen, das als bedingte Wahrscheinlichkeit formuliert ist, grafisch darstellen. Abbildung 7.11 zeigt das Bayes'sche Netzwerk für das Alarmbeispiel. Jeder Knoten im Netzwerk repräsentiert eine Variable und jede gerichtete Kante eine Aussage über die bedingte Wahrscheinlichkeit. Die Kante von Al nach J stellt beispielsweise die beiden Werte $P(J|Al)$ und $P(J|\neg Al)$ dar, die in Form einer Tabelle, der sogenannten CPT (Conditional Probability Table), angegeben werden. Die CPT eines Knotens listet alle bedingten Wahrscheinlichkeiten der Variablen des Knotens auf, die von allen Knoten abhängig sind, die durch eingehende Kanten verbunden sind.

Beim Studium des Netzwerks fragen wir uns vielleicht, warum außer den vier eingezeichneten Kanten keine weiteren Kanten enthalten sind. Die beiden Knoten Bur und Ear sind nicht verknüpft, da die Variablen unabhängig sind. Alle anderen Knoten haben einen übergeordneten Knoten, was die Argumentation etwas komplexer macht. Wir brauchen zunächst das Konzept der bedingten Unabhängigkeit.

7.4.3 Bedingte Unabhängigkeit

Analog zur Unabhängigkeit von Zufallsvariablen geben wir an

Definition 7.6 Zwei Variablen A und B heißen *bedingt unabhängig*, gegeben C wenn

$$P(A, B|C) = P(A|C) \cdot P(B|C).$$

Diese Gleichung gilt für alle Wertekombinationen für alle drei Variablen (also für die Verteilung), die wir in der Notation sehen. Wir betrachten nun im Alarmbeispiel die Knoten J und M , die den gemeinsamen Elternknoten Al haben. Wenn John und Mary

unabhängig auf einen Alarm reagieren, dann sind die beiden Variablen J und M bei gegebenem *AI* *unabhängig*, das heißt:

$$P(J, M | AI) = P(J | AI) \cdot P(M | AI).$$

Ist der Wert von *AI* bekannt, beispielsweise weil ein Alarm ausgelöst wurde, sind die Variablen J und M unabhängig (unter der Bedingung $AI = w$). Aufgrund der bedingten Unabhängigkeit der beiden Variablen J und M wird keine Kante zwischen diesen beiden Knoten hinzugefügt. Allerdings sind J und M nicht unabhängig (siehe Aufgabe 7.11 auf Seite 159).

Ganz ähnlich ist der Zusammenhang zwischen den beiden Variablen J und *Bur*, denn John reagiert nicht auf einen Einbruch, sondern auf den Alarm. Dies könnte zum Beispiel daran liegen, dass eine hohe Mauer ihm die Sicht auf Bobs Grundstück versperrt, er den Alarm aber dennoch hören kann. Somit sind J und *Bur* bei gegebenem *AI* und unabhängig

$$P(J, Bur | AI) = P(J | AI) \cdot P(Bur | AI).$$

Bei einem Alarm sind auch die Variablen J und *Ear*, M und *Bur* sowie M und *Ear* unabhängig. Für die Berechnung mit bedingter Unabhängigkeit sind die folgenden Charakterisierungen hilfreich, die der obigen Definition entsprechen:

Satz 7.5 Die folgenden Gleichungen sind paarweise äquivalent, was bedeutet, dass jede einzelne Gleichung die bedingte Unabhängigkeit für die Variablen A und B bei gegebenem C beschreibt.

$$P(A, B | C) = P(A | C) \cdot P(B | C), \quad P(A | B, C) \quad (7.16)$$

$$= P(A | C), \quad P(B | A, C) = P(B | C) \quad (7.17)$$

$$C). \quad (7.18)$$

Beweis Einerseits können wir mithilfe der bedingten Unabhängigkeit (7.16) daraus schließen

$$P(A, B, C) = P(A, B | C)P(C) = P(A | C)P(B | C)P(C).$$

Andererseits gibt uns die Produktregel

$$P(A, B, C) = P(A | B, C)P(B | C)P(C).$$

Somit ist $P(A | B, C) = P(A | C)$ äquivalent zu (7.16). Analog erhalten wir (7.18) durch Vertauschen von A und B in dieser Ableitung.

7.4.4 Praktische Anwendung

Nun wenden wir uns wieder dem Alarmbeispiel zu und zeigen, wie das Bayes-Netzwerk in Abb. 7.11 auf Seite 146 zur Argumentation genutzt werden kann. Bob interessiert sich beispielsweise für die Sensibilität seiner beiden Alarmreporter John und Mary, also für $P(J|Bur)$ und $P(M|Bur)$. Noch wichtiger sind ihm jedoch die Werte $P(Bur|J)$ und $P(Bur|M)$ sowie $P(Bur|J,M)$. Wir beginnen mit $P(J|Bur)$ und berechnen

$$P(J|Bur) = \frac{P(J,Bur)}{P(Bur)} = \frac{P(J,Bur,Al) + P(J,Bur,\neg Al)}{P(Bur)} \quad (7.19)$$

Und

$$P(J,Bur,Al) = P(J|Bur,Al)P(Al|Bur)P(Bur) = P(J|Al)P(Al|Bur)P(Bur), \quad (7.20)$$

wobei wir für die letzten beiden Gleichungen die Produktregel und die bedingte Unabhängigkeit von J und Bur bei gegebenem Al verwendet haben. Eingefügt in (7.19) erhalten wir

$$\begin{aligned} P(J|Bur) &= \frac{P(J|Al)P(Al|Bur)P(Bur) + P(J|\neg Al)P(\neg Al|Bur)P(Bur)}{P(Bur)} \\ &= P(J|Al)P(Al|Bur) + P(J|\neg Al)P(\neg Al|Bur). \end{aligned} \quad (7.21)$$

Hier fehlen $P(Al|Bur)$ und $P(\neg Al|Bur)$. Deshalb rechnen wir

$$\begin{aligned} P(Al|Bur) &= \frac{P(Al,Bur)}{P(Bur)} = \frac{P(Al,Bur,Ohr) + P(Al,Bur,\neg Ohr)}{P(Bur)} \\ &= \frac{P(Al|Bur,Ohr)P(Bur)P(Ohr) + P(Al|Bur,\neg Ohr)P(Bur)P(\neg Ohr)}{P(Bur)} \\ &= P(Al|Bur,Ohr)P(Ohr) + P(Al|Bur,\neg Ohr)P(\neg Ohr) \\ &= 0,95 \cdot 0,002 + 0,94 \cdot 0,998 = 0,94 \end{aligned}$$

sowie $P(\neg Al|Bur) = 0,06$ und fügen Sie dies in (7.21) ein, was das Ergebnis ergibt

$$P(J|Bur) = 0,9 \cdot 0,94 + 0,05 \cdot 0,06 = 0,849.$$

Analog berechnen wir $P(M|Bur) = 0,659$. Wir wissen jetzt, dass John für etwa 85 % aller Einbrüche verantwortlich ist und Mary für etwa 66 % aller Einbrüche. Die Wahrscheinlichkeit, die beide nennen, wird aufgrund der bedingten Unabhängigkeit wie folgt berechnet

$$P(J,M|Bur) = P(J|Bur)P(M|Bur) = 0,849 \cdot 0,659 = 0,559.$$

Interessanter ist jedoch die Wahrscheinlichkeit eines Anrufs von John oder Mary

$$\begin{aligned}
 P(J \vee M | Bur) &= P(\neg(\neg J, \neg M) | Bur) = 1 - P(\neg J, \neg M | Bur) \\
 &= 1 - P(\neg J | Bur)P(\neg M | Bur) = 1 - 0,051 = 0,948.
 \end{aligned}$$

Somit wird Bob bei etwa 95 % aller Einbrüche benachrichtigt. Um nun $P(Bur | J)$ zu berechnen, wenden wir die Bayes-Formel an, die uns ergibt

$$P(Bur | J) = \frac{P(J | Bur)P(Bur)}{P(J)} = \frac{0,849 \cdot 0,001}{0,052} = 0,016.$$

Offenbar sind nur etwa 1,6 % aller Anrufe von John tatsächlich auf einen Einbruch zurückzuführen. Da die Wahrscheinlichkeit von Fehlalarmen bei Mary fünfmal geringer ist, haben wir mit $P(Bur | M) = 0,056$ ein deutlich höheres Vertrauen bei einem Anruf von Mary. Bob sollte sich nur ernsthafte Sorgen um sein Zuhause machen, wenn beide anrufen, denn $P(Bur | J, M) = 0,284$ (siehe Übung 7.11 auf Seite 159).

In (7.21) auf Seite 148 haben wir mit gezeigt

$$P(J | Bur) = P(J | AI)P(AI | Bur) + P(J | \neg AI)P(\neg AI | Bur)$$

wie wir eine neue Variable „einschieben“ können. Diese Beziehung gilt im Allgemeinen für zwei Variablen A und B bei Einführung einer zusätzlichen Variablen C und wird als *Konditionierung bezeichnet*:

$$P(A | B) = \sum_c P(A | B, C = c)P(C = c | B).$$

Wenn außerdem A und B bei gegebenem C bedingt unabhängig sind, vereinfacht sich diese Formel zu

$$P(A | B) = \sum_c P(A | C = c)P(C = c | B).$$

7.4.5 Software für Bayesianische Netzwerke

Am Beispiel Alarm geben wir eine kurze Einführung in zwei Tools. Das System PIT ist uns bereits bekannt. Die Werte aus den CPTs geben wir in PIT-Syntax in das Online-Eingabefenster unter www.pit-systems.de ein. Nach der in Abb. 7.12 auf Seite 150 dargestellten Eingabe erhalten wir die Antwort:

$$P([Einbruch=t] | [John=t] \text{ AND } [Mary=t]) = 0,2841.$$

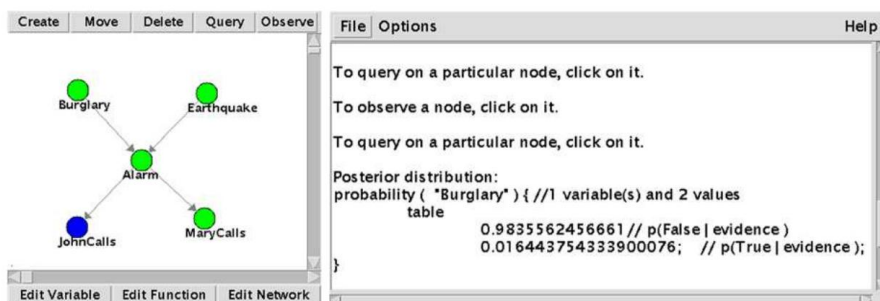
Obwohl PIT kein klassisches Bayes'sches Netzwerktool ist, kann es beliebige bedingte Wahrscheinlichkeiten und Abfragen als Eingabe verwenden und korrekte Ergebnisse berechnen. Es kann gezeigt werden [Sch96], dass das MaxEnt-Prinzip bei Eingabe von CPTs oder äquivalenten Regeln die gleichen bedingten Unabhängigkeiten und damit auch die gleichen Antworten impliziert wie ein Bayesianisches Netzwerk. Bayesianische Netzwerke sind somit ein Sond

```

1 var Alarm{t,f}, Einbruch{t,f}, Erdbeben{t,f}, John{t,f}, Mary{t,f};
2
3 P([Erdbeben=t] = 0,002; 4 P([Einbruch=t])
= 0,001; 5 P([Alarm=t] | [Einbruch=t]
UND [Erdbeben=t]) = 0,95; 6 P([Alarm=t] | [Einbruch=t] UND [Erdbeben=f]) = 0,94; 7
P([Alarm=t] | [Einbruch=f] UND [Erdbeben=t]) = 0,29; 8 P([Alarm=t] | [Einbruch=f]
UND [Erdbeben=f]) = 0,001; 9 P([John=t] | [Alarm=t]) = 0,90; 10 P([John=t] | [Alarm=f])
= 0,05; 11 P([Mary=t] | [Alarm=t]) = 0,70; 12 P([Mary=t] | [Alarm=f]) = 0,01; 13 14
QP([Einbruch=t] | [John=t] AND [Mary=t]);

```

Abb. 7.12 PIT-Eingabe für das Alarmbeispiel

Abb. 7.13 Die Benutzeroberfläche von JavaBayes: *Links* der grafische Editor und *rechts* die Konsole, in der die Antworten als Ausgabe ausgegeben werden

Als nächstes betrachten wir JavaBayes [Coz98], ein klassisches System, das ebenfalls frei im Internet verfügbar ist und über die in Abb. 7.13 dargestellte grafische Oberfläche verfügt. Mit dem grafischen Netzwerkeditor können Knoten und Kanten manipuliert und die Werte in den CPTs bearbeitet werden. Darüber hinaus können mit „Observe“ die Werte von Variablen zugewiesen und mit „Query“ die Werte anderer Variablen abgerufen werden. Die Antworten auf Abfragen erscheinen dann im Konsolenfenster.

Das professionelle, kommerzielle System Hugin ist deutlich leistungsfähiger und komfortabler. Beispielsweise kann Hugin neben diskreten Variablen auch kontinuierliche Variablen verwenden. Es kann auch Bayes'sche Netzwerke lernen, also das Netzwerk vollautomatisch aus statistischen Daten generieren (siehe Abschn. 8.5).

7.4.6 Entwicklung Bayes'scher Netzwerke

Ein kompaktes Bayes-Netzwerk ist sehr übersichtlich und für den Leser deutlich informativer als eine vollständige Wahrscheinlichkeitsverteilung. Darüber hinaus wird viel weniger benö

Erinnerung. Für die Variablen $v_1, \dots, v_n$ mit $|v_1|, \dots, |v_n|$ jeweils unterschiedliche Werte, die Verteilung hat insgesamt

$$\prod_{i=1}^n (|v_i| + 1)$$

unabhängige Einträge. Im Alarmbeispiel sind alle Variablen binär. Somit gilt für alle Variablen $|v_i| = 2$, und die Verteilung hat $25 + 1 = 31$ unabhängige Einträge. Um die Anzahl der unabhängigen Einträge für das Bayes-Netzwerk zu berechnen, muss die Gesamtzahl aller Einträge aller CPTs ermittelt werden. Für einen Knoten v_i mit k_i Elternknoten $e_{i1}, \dots, e_{ik_i}$, das zugehörige CPT hat

$$\prod_{j=1}^{k_i} (|v_i| + 1)^{|e_{ij}|}$$

Einträge. Dann haben alle CPTs im Netzwerk zusammen

$$\prod_{i=1}^n \prod_{j=1}^{k_i} (|v_i| + 1)^{|e_{ij}|} \quad (7.22)$$

Einträge. Für das Alarmbeispiel lautet das Ergebnis dann

$$2 + 2 + 4 + 1 + 1 = 10$$

unabhängige Einträge, die das Netzwerk eindeutig beschreiben. Der Vergleich der Speicherkomplexität zwischen der Vollverteilung und dem Bayes'schen Netzwerk wird klarer, wenn wir annehmen, dass alle n Variablen die gleiche Anzahl b an Werten haben und jeder Knoten k Elternknoten hat. Dann kann (7.22) vereinfacht werden und alle CPTs zusammen haben $n(b + 1)^b$ Einträge. Die vollständige Verteilung enthält $b^n + 1$ Einträge. Ein Signifikant Überhöhungsgewinn wird nur dann erzielt, wenn die durchschnittliche Anzahl der übergeordneten Knoten viel kleiner ist als die Anzahl der Variablen. Dies bedeutet, dass die Knoten nur lokal. Durch die lokale Anbindung wird das Netzwerk modularisiert, was – wie im Software-Engineering – zu einer Reduzierung der Komplexität führt. Im Alarmbeispiel trennt der Alarmknoten die Knoten *Bur* und *Ear* von den Knoten *J* und *M*. Dies können wir auch im LEXMED-Beispiel deutlich sehen.

¹⁵Für den Fall eines Knotens ohne Vorfahren ist das Produkt in dieser Summe leer. Dafür ersetzen wir den Wert 1, da die CPT für Knoten ohne Vorfahren mit ihrer A-priori-Wahrscheinlichkeit genau einen Wert enthält.

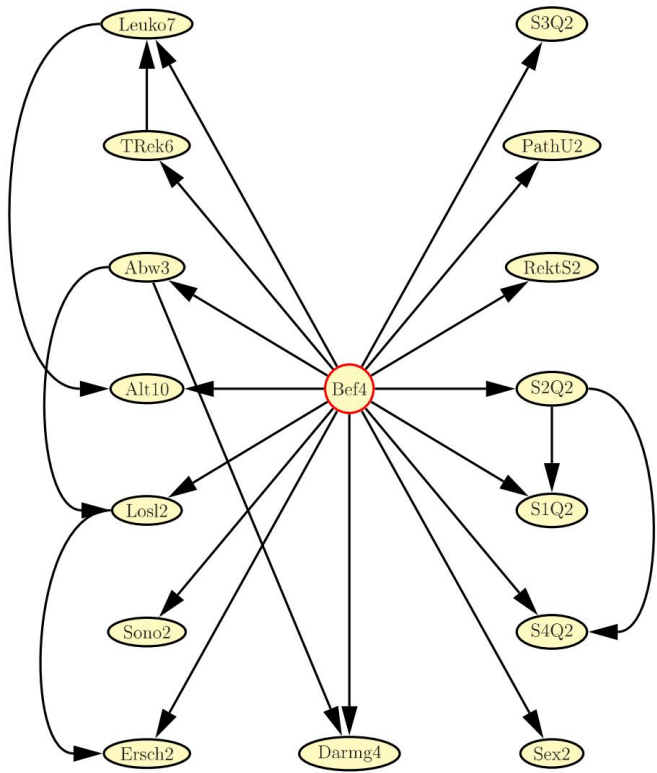


Abb. 7.14 Bayesianisches Netzwerk für die LEXMED- Anwendung

LEXMED als Bayesianisches Netzwerk

Das in Abschn. beschriebene LEXMED- System. 7.3 kann auch als Bayes'sches Netzwerk modelliert werden. Indem man die äußeren, dünn gezeichneten Linien gerichtet macht (sie mit Pfeilen versehen), kann der Unabhängigkeitsgraph in Abb. 7.8 auf Seite 137 als Bayes'sches Netzwerk interpretiert werden. Das resultierende Netzwerk ist in Abb. 7.14 dargestellt .

In Abschn. 7.3.2 Die Größe der Verteilung für LEXMED wurde berechnet als Wert 20 643 839. Das Bayes'sche Netzwerk hingegen kann mit nur 521 Werten vollständig beschrieben werden. Dieser Wert kann ermittelt werden, indem die Variablen aus Abb. 7.14 in (7.22) auf Seite 151 eingegeben werden. In der Reihenfolge (Leuko, TRek, Gua, Age, Reb, Sono, Tapp, BowS, Sex, P4Q, P1Q, P2Q, RecP , Urin, P3Q, Diag4) berechnen wir

$$\begin{aligned} & \sum_{i=1}^N \sum_{k_i} (|v_i| - 1) \cdot |e_{ij}| = 6 \cdot 6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 7 \cdot 4 + 1 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 2 \cdot 4 = 1 \\ & \quad + 3 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \\ & \quad + 1 \cdot 4 + 1 = 521. \end{aligned}$$

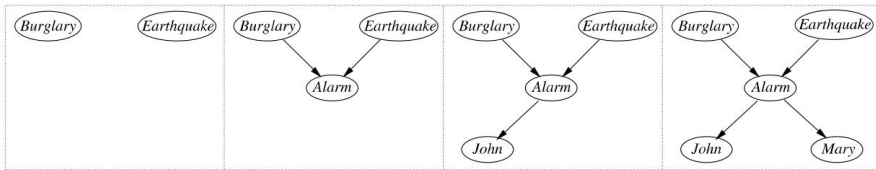


Abb. 7.15 Schrittweiser Aufbau des Alarmnetzwerks unter Berücksichtigung der Kausalität

Dieses Beispiel zeigt, dass es praktisch unmöglich ist, eine vollständige Distribution für reale Anwendungen zu erstellen. Ein Bayes-Netzwerk mit 22 Kanten und 521 Wahrscheinlichkeitswerten ist dagegen noch beherrschbar.

Kausalität und Netzwerkstruktur

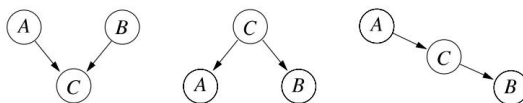
Der Aufbau eines Bayes'schen Netzwerks erfolgt normalerweise in zwei Schritten.

1. **Entwurf der Netzwerkstruktur:** Dieser Schritt wird in der Regel manuell durchgeführt und im Folgenden beschrieben.
2. **Eingabe der Wahrscheinlichkeiten in die CPTs:** Bei vielen Variablen ist die manuelle Eingabe der Werte sehr mühsam. Wenn (wie zum Beispiel bei LEXMED) eine Datenbank verfügbar ist, kann dieser Schritt automatisiert werden, indem die CPT-Einträge durch Zählen von Häufigkeiten geschätzt werden.

Wir beschreiben nun den Aufbau des Netzwerks am Alarmbeispiel (siehe Abb. 7.15). Zu Beginn kennen wir die beiden Ursachen *Einbruch* und *Erdbeben* und die beiden Symptome *John* und *Mary*. Da John und Mary jedoch nicht direkt auf einen Einbrecher oder ein Erdbeben reagieren, sondern nur auf den Alarm, ist es angebracht, dies als zusätzliche Variable hinzuzufügen, die von Bob nicht beobachtbar ist. Der Prozess des Hinzufügens von Kanten beginnt bei den Ursachen, also bei den Variablen, die keine übergeordneten Knoten haben. Zuerst wählen wir *Einbruch* und als nächstes *Erdbeben*. Jetzt müssen wir prüfen, ob *Erdbeben* unabhängig von *Einbruch* ist. Dies ist gegeben und daher wird kein Vorteil von *Einbruch* zu *Erdbeben* hinzugefügt. Da „Alarm“ direkt von „Einbruch“ und „Erdbeben“ abhängt, werden diese Variablen als Nächstes ausgewählt und eine Kante von „Einbruch“ und „Erdbeben“ zu „Alarm“ hinzugefügt. Dann wählen wir *John*. Da *Alarm* und *John* nicht unabhängig sind, wird eine Kante von Alarm zu John hinzugefügt. Dasselbe gilt auch für *Maria*. Nun müssen wir prüfen, ob *John* bei *Einbruchalarm* bedingt unabhängig ist. Ist dies nicht der Fall, muss eine weitere Kante von *Einbruch* zu *John* eingefügt werden. Wir müssen auch prüfen, ob Kanten von *Erdbeben* zu *John* und von *Einbruch* oder *Erdbeben* zu *Mary* benötigt werden. Aufgrund der bedingten Unabhängigkeit sind diese vier Kanten nicht notwendig. Kanten zwischen *John* und *Mary* sind auch unnötig, da *John* und *Mary* bei *Alarm* bedingt unabhängig sind.

Die Struktur des Bayes'schen Netzwerks hängt stark von der gewählten Variablenreihenfolge ab. Wenn die Reihenfolge der Variablen so gewählt wird, dass sie den Kausalzusammenhang widerspiegelt, beginnend bei den Ursachen und fortschreitend zu den Diagnosevariablen, dann wird das Ergebnis ein einfaches Netzwerk sein. Andernfalls kann das Netzwerk deutlich mehr Kanten enthalten. Solche nicht-kausalen Netzwerke sind oft sehr schwer zu verstehen und haben eine höhere Relevanz

Abb. 7.16 Es gibt keine Kante zwischen A und B, wenn sie unabhängig (*links*) oder bedingt unabhängig (*Mitte, rechts*) sind.



Komplexität für das Denken. Zum besseren Verständnis kann der Leser auf Übung 7.11 auf Seite 159 verweisen .

7.4.7 Semantik Bayes'scher Netzwerke

Wie wir im vorherigen Abschnitt gesehen haben, wird einem Bayes'schen Netzwerk zwischen zwei Variablen A und B keine Kante hinzugefügt, wenn A und B unabhängig oder bedingt unabhängig sind und eine dritte Variable C gegeben ist. Diese Situation ist in Abb. 7.16 dargestellt.

Wir fordern nun, dass das Bayes'sche Netzwerk keine Zyklen hat, und gehen davon aus, dass die Variablen so nummeriert sind, dass keine Variable eine niedrigere Nummer hat als jede Variable, die ihr vorangeht. Dies ist immer dann möglich, wenn das Netzwerk keine Zyklen hat.¹⁶ Dann gilt, wenn alle bedingten Unabhängigkeiten verwendet werden

$$P(X_n | X_1, \dots, X_{n-1}) = P(X_n | \text{Eltern}(X_n)).$$

Diese Gleichung ist somit eine Aussage, dass eine beliebige Variable X_i in einem Bayes'schen Netzwerk angesichts ihrer Eltern bedingt unabhängig von ihren Vorfahren ist. Der etwas allgemeinere Satz, der in Abb. 7.17 auf Seite 155 dargestellt ist, lässt sich kompakt formulieren als

Satz 7.6 Ein Knoten in einem Bayes'schen Netzwerk ist *aufgrund seiner Eltern bedingt unabhängig von allen Nicht-Nachfolgeknoten* .

Jetzt können wir die Kettenregel ((7.1) auf Seite 120) stark vereinfachen:

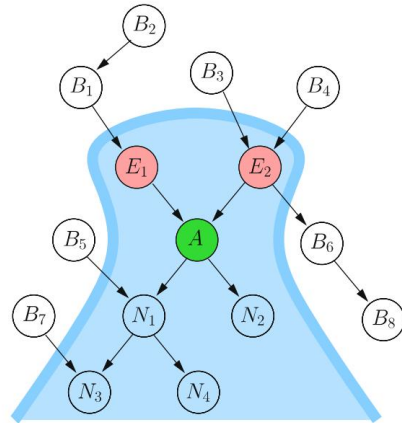
$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) = \prod_{i=1}^n P(X_i | \text{Eltern}(X_i)). \quad (7.23)$$

Mit dieser Regel könnten wir beispielsweise (7.20) direkt auf Seite 148 schreiben

$$P(J, Bur, AI) = P(J | AI)P(AI | Bur)P(Bur).$$

¹⁶Wenn zum Beispiel drei Knoten X_1, X_2, X_3 einen Kreis bilden, dann gibt es die Kanten (X_1, X_2) , (X_2, X_3) und (X_3, X_1) .

Abb. 7.17 Beispiel für bedingte Unabhängigkeit in einem Bayes'schen Netzwerk. Sind die Elternknoten E_1 und E_2 gegeben, dann sind alle Nicht-Nachfolgeknoten $B_1, \dots, B_8$ unabhängig von A



Wir kennen jetzt die wichtigsten Konzepte und Grundlagen des Bayes'schen Netzes funktioniert. Fassen wir sie zusammen [Jen01]:

Definition 7.7 Ein Bayes'sches Netzwerk ist definiert

durch: • Eine Menge von Variablen und eine Menge gerichteter Kanten zwischen diesen Variablen. • Jede Variable hat endlich viele mögliche Werte. • Die Variablen bilden zusammen mit den Kanten einen gerichteten azyklischen Graphen.

Ein DAG ist ein Graph ohne Zyklen, also ohne Pfade der Form $(A, \dots, A)$.

• Für jede Variable A die CPT (d. h. die Tabelle der bedingten Wahrscheinlichkeiten). $P(A|\text{Eltern}(A))$ ist gegeben.

Zwei Variablen A und B heißen *bedingt unabhängig* bei gegebenem C , wenn $P(A, B|C) = P(A|C) \cdot P(B|C)$ oder äquivalent, wenn $P(A|B, C) = P(A|C)$.

Neben den Grundregeln der Wahrscheinlichkeitsberechnung gelten auch die folgenden Regeln: $P(B|A) \cdot P(A)$

Satz von Bayes: $P(A|B) = P(B) \cdot \frac{P(A|B)}{P(A)}$

Marginalisierung: $P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)$

Konditionierung: $P(A|B) = P(A|B, C = c)P(C = c|B)$

Eine Variable in einem Bayes'schen Netzwerk ist aufgrund ihrer übergeordneten Variablen bedingt unabhängig von allen Nicht-Nachfolgevariablen. Wenn $X_1, \dots, X_{n-1}$ keine Nachfolger von X_n sind, gilt $P(X_n|X_1, \dots, X_{n-1}) = P(X_n|\text{Parents}(X_n))$. Diese Bedingung muss beim Aufbau eines Netzwerks berücksichtigt werden.

Beim Aufbau eines Bayes'schen Netzwerks sollten die Variablen nach Kausalität geordnet werden. Zuerst die Ursachen, dann die versteckten Variablen und zuletzt die Diagnosevariablen.

Kettenregel: $P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Eltern}(X_i))$

In [Pea88] und [Jen01] wird der Begriff d-Separation für Bayes'sche Netze eingeführt, woraus ein Theorem ähnlich zu Theorem 7.6 auf Seite 154 folgt. Wir verzichten auf die Einführung dieses Begriffs und kommen dadurch zu einer etwas einfacheren, wenn auch theoretisch nicht ganz so sauberen Darstellung.

7.5 Zusammenfassung

In einer Weise, die den anhaltenden, anhaltenden Trend zu probabilistischen Systemen widerspiegelt, haben wir probabilistische Logik für das Denken mit unsicherem Wissen eingeführt. Nach der Einführung der Sprache – um Wahrscheinlichkeiten oder Wahrscheinlichkeitsintervalle erweiterte Aussagenlogik – wählten wir den natürlichen, wenn auch ungewöhnlichen Ansatz über die Methode der maximalen Entropie als Einstiegspunkt und zeigten, wie wir mit dieser Methode nichtmonotones Denken modellieren können. Bayesianische Netzwerke wurden dann als Sonderfall der MaxEnt-Methode eingeführt.

Warum sind Bayesianische Netzwerke ein Sonderfall von MaxEnt? Beim Aufbau eines Bayes'schen Netzwerks werden Annahmen zur Unabhängigkeit getroffen, die für die MaxEnt-Methode nicht erforderlich sind. Darüber hinaus müssen beim Aufbau eines Bayes'schen Netzwerks alle CPTs vollständig ausgefüllt werden, damit eine vollständige Wahrscheinlichkeitsverteilung entsteht. Andernfalls ist die Argumentation eingeschränkt oder unmöglich. Mit MaxEnt hingegen kann der Entwickler sämtliches Wissen, das ihm zur Verfügung steht, in Form von Wahrscheinlichkeiten formulieren. Anschließend vervollständigt MaxEnt das Modell und generiert die Verteilung. Auch wenn (beispielsweise bei der Befragung eines Experten) nur vage Aussagen vorliegen, kann dies geeignet modelliert werden. Eine Aussage wie „Ich bin mir ziemlich sicher, dass A wahr ist.“ kann beispielsweise mit $P(A) \in [0,6, 1]$ als Wahrscheinlichkeitsintervall modelliert werden. Beim Aufbau eines Bayes'schen Netzwerks muss, ggf. durch Schätzung, ein konkreter Wert für die Wahrscheinlichkeit angegeben werden. Dies bedeutet jedoch, dass der Experte oder der Entwickler Ad-hoc-Informationen in das System einpflegt. Ein weiterer Vorteil von MaxEnt ist die Möglichkeit, (fast) beliebige Aussagen zu formulieren. Für Bayes'sche Netzwerke müssen die CPTs gefüllt sein.

Die Freiheit, die der Entwickler bei der Modellierung mit MaxEnt hat, kann (insbesondere für einen Anfänger) von Nachteil sein, da im Gegensatz zum Bayesianischen Ansatz nicht unbedingt klar ist, welches Wissen modelliert werden soll. Bei der Modellierung mit Bayes'schen Netzen ist die Vorgehensweise ganz klar: Entsprechend der kausalen Abhängigkeiten, von den Ursachen bis zu den Wirkungen, wird durch Prüfung der bedingten Unabhängigkeit eine Kante nach der anderen in das Netz eingetragen.¹⁷ Am Ende werden alle CPTs mit Werten ausgefüllt.

Folgende interessante Kombinationen der beiden Methoden sind jedoch möglich: Wir bauen zunächst ein Netzwerk nach der Bayes'schen Methodik auf, geben alle Kanten entsprechend ein und füllen dann die CPTs mit Werten. Sollten bestimmte Werte für die CPTs nicht verfügbar sein, können sie durch Intervalle oder andere probabilistische Logikformeln ersetzt werden. Natürlich gibt es ein solches Netzwerk – oder besser: ein Regelwerk – nicht

¹⁷Das ist auch nicht immer ganz so einfach.

hat nicht mehr die spezielle Semantik eines Bayes'schen Netzwerks. Anschließend muss es von einem MaxEnt-System verarbeitet und vervollständigt werden.

Die Möglichkeit, MaxEnt mit beliebigen Regelsätzen zu verwenden, hat jedoch einen Nachteil. Ähnlich wie in der Logik können solche Regelsätze inkonsistent sein. Beispielsweise sind die beiden Regeln $P(A) = 0,7$ und $P(A) = 0,8$ inkonsistent. Während beispielsweise das MaxEnt-System PIT die Inkonsistenz erkennen kann, kann es keinen Hinweis darauf geben, wie das Problem behoben werden kann.

Wir stellten das medizinische Expertensystem LEXMED vor, eine klassische Anwendung zur Argumentation mit unsicherem Wissen, und zeigten, wie es mithilfe von MaxEnt und Bayesianischen Netzwerken modelliert und implementiert werden kann und wie diese Tools das etablierte, aber zu schwache lineare Scoring ersetzen können Systeme, die in der Medizin verwendet werden.

Im Beispiel von LEXMED haben wir gezeigt, dass es möglich ist, ein Expertensystem für das Denken unter Unsicherheit aufzubauen, das in der Lage ist, Wissen aus den Daten in einer Datenbank zu entdecken (zu lernen). Weitere Einblicke in die Lernmethoden Bayes'scher Netzwerke geben wir in Kap. 8, nachdem die notwendigen Grundlagen für maschinelles Lernen gelegt wurden.

Heute ist das Bayesianische Denken ein großes, eigenständiges Feld, das wir hier nur kurz beschreiben können. Auf die Behandlung kontinuierlicher Variablen haben wir komplett verzichtet. Für den Fall normalverteilter Zufallsvariablen gibt es Verfahren und Systeme. Bei beliebigen Verteilungen ist jedoch der Rechenaufwand ein großes Problem. Neben den gerichteten Netzwerken, die stark auf Kausalität basieren, gibt es auch ungerichtete Netzwerke. Damit verbunden ist eine Diskussion über die Bedeutung und den Nutzen von Kausalität in Bayes'schen Netzwerken. Der interessierte Leser wird auf hervorragende Lehrbücher wie [Pea88, Jen01, Whi96, DHS01] sowie auf die Tagungsberichte der Jahreskonferenz der Association for Uncertainty in Artificial Intelligence (AUAI) verwiesen (www.auai.org).

7.6 Übungen

Aufgabe 7.1 Beweisen Sie den Satz aus Satz 7.1 auf Seite 117.

Übung 7.2 Der Hobbygärtner Max möchte seine jährliche Erbsenernte statistisch analysieren. Für jede Erbsenschote, die er pflückt, misst er deren Länge x_i in Zentimetern und ihr Gewicht y_i in Gramm. Er teilt die Erbsen in zwei Klassen ein, die guten und die schlechten (leeren Schoten). Die gemessenen Daten (x_i, y_i) sind

	x 122334456 und 234455666	x 4667 und
gute Erbsen:		2233
		schlechte Erbsen:

18In Abschn. 8.6 und in Übung 8.16 auf Seite 219 werden wir zeigen, dass die Scores äquivalent zum Spezialfall naive Bayes sind, also zur Annahme, dass alle Symptome angesichts der Diagnose bedingt unabhängig sind.

- (a) Berechnen Sie aus den Daten die Wahrscheinlichkeiten $P(y > 3 \mid \text{Klasse} = \text{gut})$ und $P(y \leq 3 \mid \text{Klasse} = \text{gut})$. Verwenden Sie dann die Bayes-Formel, um $P(\text{Klasse} = \text{gut} \mid y > 3)$ und $P(\text{Klasse} = \text{gut} \mid y \leq 3)$.
- (b) Welche der in Teilaufgabe (a) berechneten Wahrscheinlichkeiten widerspricht der Aussage? „Alle guten Erbsen sind schwerer als 3 Gramm“?

Übung 7.3 Sie sollen das Nachmittagswetter mithilfe einiger einfacher Vorhersagen vorhersagen. Wetterwerte vom Morgen dieses Tages. Die klassische Wahrscheinlichkeitsrechnung. Hierzu ist ein vollständiges Modell erforderlich, das in der folgenden Tabelle aufgeführt ist.

Himmel	Bar	Prec P (Sky, Bar, Prec)
Klar	Steigende	0,40
Klar	Trockenheit	Steigende Regenfälle 0,07
Klar	Trockenfallen	0,08
Klar	Fallender Regen	0,10
Bewölkt	Steigend	Trocken 0,09
Bewölkt	Steigend	Regen 0,11
Bewölkt	fallend	trocken 0,03

Himmel: Der Himmel ist klar oder bewölkt am Morgen
Balken: Barometer steigend bzw am Morgen fallen
Prec: Regen oder Trockenheit im Nachmittag

- (a) Wie viele Ereignisse enthält die Verteilung für diese drei Variablen?
 (b) Berechnen Sie $P(\text{Prec} = \text{trocken}, \text{Himmel} = \text{klar}, \text{Balken} = \text{steigend})$.
 (c) Berechnen Sie $P(\text{Prec} = \text{Regen} \mid \text{Sky} = \text{bewölkt})$.
 (d) Was würden Sie tun, wenn die letzte Zeile in der Tabelle fehlen würde?

Übung 7.4 In einer Fernsehquizshow muss der Teilnehmer zwischen drei wählen verschlossene Türen. Hinter einer Tür wartet der Preis: ein Auto. Hinter beiden Türen sind Ziegen. Der Teilnehmer wählt eine Tür, z. B. Nummer eins. Der Gastgeber, der weiß, wo das Auto ist, öffnet eine andere Tür, z. B. Nummer drei, und eine Ziege erscheint. Der Teilnehmer hat nun die Möglichkeit, zwischen den beiden verbleibenden zu wählen Türen (eine und zwei). Was ist aus seiner Sicht die bessere Wahl? Bleiben mit die Tür, die er ursprünglich gewählt hatte, oder zu der anderen geschlossenen Tür wechseln?

Übung 7.5 Zeigen Sie dies mithilfe der Lagrange-Multiplikatormethode, ohne explizit Einschränkungen stellt die Gleichverteilung $p_1 = p_2 = \dots = p_n = 1/n$ die maximale Entropie dar. Vergessen Sie nicht die implizit allgegenwärtige Einschränkung $p_1 + p_2 + \dots + p_n = 1$. Wie können wir dasselbe Ergebnis mithilfe der Indifferenz zeigen?

Aufgabe 7.6 Verwenden Sie das PIT-System (www.pit-systems.de), um den MaxEnt zu berechnen. Lösung für $P(B)$ unter der Randbedingung $P(A) = \bar{y}$ und $P(B|A) = \bar{y}$. Welchen Nachteil der PIT im Vergleich zur Berechnung per Hand fällt Ihnen hier auf?

Übung 7.7 Berechnen Sie $P(B)$ anhand der Randbedingungen $P(A) = \bar{y}$ und $P(A \mid B) = \bar{y}$ manuell mit der MaxEnt-Methode. Verwenden Sie PIT, um Ihre Lösung zu überprüfen.

Ü Aufgabe 7.8 Gegeben seien die Nebenbedingungen aus (7.10), (7.11), (7.12): $p_1 + p_2 = \bar{y}$, $p_1 + p_3 = \bar{y}$, $p_1 + p_2 + p_3 + p_4 = 1$. Zeigen Sie, dass $p_1 = \bar{y}\bar{y}$, $p_2 = \bar{y}(1 - \bar{y}\bar{y})$, $p_3 = \bar{y}(1 - \bar{y}\bar{y})$, $p_4 = (1 - \bar{y}\bar{y})(1 - \bar{y}\bar{y})$ stellt das Entropiemaximum unter diesen Einschränkungen dar.

Ü Übung 7.9 Ein probabilistischer Algorithmus berechnet die Wahrscheinlichkeit p , dass es sich bei einer eingehenden E-Mail um Spam handelt. Um die E-Mails in die Klassen „Löschen“ und „Lesen“ zu klassifizieren, wird

anschließend eine Kostenmatrix auf das Ergebnis angewendet. (a) Geben Sie eine Kostenmatrix (2×2 -Matrix) für den Spamfilter an. Gehen Sie hier davon aus, dass das Löschen einer E-Mail den Benutzer 10 Cent kostet, während der Verlust einer E-Mail 10 Dollar kostet (vergleichen Sie dies mit Beispiel 1.1 auf Seite 11 und Übung 1.7 auf Seite 14). (b) Zeigen Sie, dass für den Fall einer 2×2 -Matrix die Anwendung der Kostenmatrix äquivalent zur Anwendung eines Schwellenwerts auf die Spam-Wahrscheinlichkeit ist, und bestimmen Sie den Schwellenwert.

Aufgabe 7.10 Gegeben sei ein Bayes-Netzwerk mit den drei Binärvariablen A,B,C und $P(A) = 0,2$, $P(B) = 0,9$ sowie der unten gezeigten CPT:

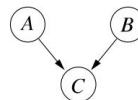
(a) Berechnen Sie $P(A|B)$.

(b) Berechnen Sie $P(C|A)$.

ABP (C) wf 0,1

ww 0,2

fw 0,9 ff 0,4



Aufgabe 7.11 Berechnen Sie für das Alarmbeispiel (Beispiel 7.7 auf Seite 145) die folgenden bedingten Wahrscheinlichkeiten: (a) Berechnen Sie

die A-priori-Wahrscheinlichkeiten $P(A)$, $P(J)$, $P(M)$. (b) Berechnen Sie

$P(M|Bur)$ unter Verwendung der Produktregel, der Marginalisierung, der Kettenregel und bedingte Unabhängigkeit. (c)

Verwenden Sie die Bayes-Formel, um $P(Bur|M)$ zu

berechnen. (d) Berechnen Sie $P(A|J,M)$ und $P(Bur|$

$J,M)$. (e) Zeigen Sie, dass die Variablen J und M nicht unabhängig

sind. (f) Überprüfen Sie alle Ihre Ergebnisse mit JavaBayes und mit PIT (siehe [Ert11] für Demo-Programme).

(g) Entwerfen Sie ein Bayes'sches Netzwerk für das Alarmbeispiel, jedoch mit der geänderten Variablenreihenfolge M, A, Ear, Bur, J . Gemäß der Semantik Bayes'scher Netzwerke müssen nur die notwendigen Kanten eingezeichnet werden. (Hinweis: Die hier angegebene Variablenreihenfolge stellt KEINE Kausalität dar. Daher wird es schwierig sein, bedingte Unabhängigkeiten intuitiv zu bestimmen.)

(h) Im ursprünglichen Bayes'schen Netzwerk des Alarmbeispiels wurden die Erdbebenknoten entfernt. Welche CPTs ändert sich dadurch? (Warum gerade diese?)

(i) Berechnen Sie die CPT des Alarmknotens im neuen Netzwerk.

Aufgabe 7.12 Es soll ein Diagnosesystem für ein dynamobetriebenes Fahrradlicht unter Verwendung eines Bayes'schen Netzwerks erstellt werden. Die Variablen in der folgenden Tabelle sind angegeben.

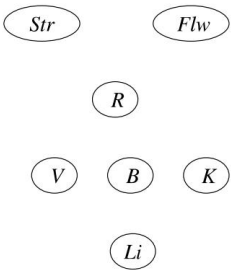
Abk.	Das bedeutet,	Werte
	dass <i>Li</i> Light im	t/f
<i>Str</i>	Straßenzustand ist	<i>trocken, nass, schneebedeckt</i>
<i>Flw</i>	Dynamo-Schwungrad verschlissen t/f	
<i>R</i>	Dynamo gleitend	t/f
<i>V</i>	Dynamo zeigt Spannung t/f an	
<i>B</i>	Glühbirne ok t/f	
<i>K</i>	Kabel ok t/f	

Die folgenden Variablen sind paarweise unabhängig:
Str, *Flw*, *B*, *K*. Weiterhin: (R, B), (R, K), (V, B),
(V, K) sind unabhängig und die folgende Gleichung
hält:

$P(Li|V, R) = P(Li|V)$

$P(V|R, Str) = P(V|R)$

$P(V/R, Flw) = P(V|R)$



- (a) Zeichnen Sie alle Kanten in den Graphen ein (unter Berücksichtigung der Kausalität).
- (b) Tragen Sie alle fehlenden CPTs in die Grafik (Tabelle der bedingten Wahrscheinlichkeiten) ein. Geben Sie für die Wahrscheinlichkeiten frei plausible Werte ein.
- (c) Zeigen Sie, dass das Netzwerk keine Kante enthält (Str, Li).
- (d) Berechnen Sie $P(V|Str = schneebedeckt)$.

VBK $P(Li)$
ttt 0,99
ttf 0,01
t ft 0,01
t ff 0,001
ft t 0,3
ft f 0,005
fft 0,005
fff 0

Maschinelles Lernen und Data Mining

8

Wenn wir KI wie in Elaine Richs Buch [Ric83] definieren:

Künstliche Intelligenz ist die Untersuchung, wie man Computer dazu bringt, bestimmte Dinge zu tun
Moment, die Menschen sind besser.

und wenn man bedenkt, dass die Lernfähigkeit des Computers diesem besonders unterlegen ist
des Menschen, dann ist die Erforschung von Lernmechanismen und die Entwicklung maschineller
Lernalgorithmen einer der wichtigsten Zweige
der KI.

Auch aus Sicht der Software besteht Bedarf an maschinellem Lernen
Entwickler, der beispielsweise das Verhalten eines autonomen Roboters programmiert. Der
Die Struktur des intelligenten Verhaltens kann so komplex werden, dass es sehr schwierig ist
Selbst mit modernen Hochsprachen wie PROLOG und Python ist es sogar unmöglich, sie annähernd
optimal zu programmieren.¹ Es kommen sogar Algorithmen des maschinellen Lernens zum Einsatz
heute Roboter auf ähnliche Weise zu programmieren, wie Menschen lernen (siehe Kap. 10 bzw
[BCDS08, RGH+06]), oft in einer hybriden Mischung aus programmiert und gelernt
Verhalten.

Die Aufgabe dieses Kapitels besteht darin, die wichtigsten Algorithmen des maschinellen Lernens
und ihre Anwendungen zu beschreiben. Das Thema wird in diesem Abschnitt vorgestellt und anschließend verfolgt
durch wichtige grundlegende Lernalgorithmen in den nächsten Abschnitten. Parallel dazu werden
Theorie und Terminologie aufgebaut. Das Kapitel wird mit einer Zusammenfassung abgeschlossen
und Überblick über die verschiedenen Algorithmen und ihre Anwendungen. Wir beschränken uns in
diesem Kapitel auf überwachte und unüberwachte Lernalgorithmen. Als wichtige Klasse lernender
Algorithmen werden neuronale Netze in Kap. behandelt. 9.
Aufgrund seiner besonderen Stellung und wichtigen Rolle für autonome Roboter, Verstärkung
Auch zum Lernen wird es ein eigenes Kapitel geben (Kap. 10).

Was ist Lernen? Erlernen des Wortschatzes einer Fremdsprache oder einer technischen Sprache
Begriffe zu lernen oder sich sogar ein Gedicht zu merken, kann für viele Menschen schwierig sein. Für Computer,

¹Python ist eine moderne Skriptsprache mit sehr gut lesbarer Syntax, leistungsstarken Datentypen und
umfangreichen Standardbibliotheken, die zu diesem Zweck verwendet werden können.

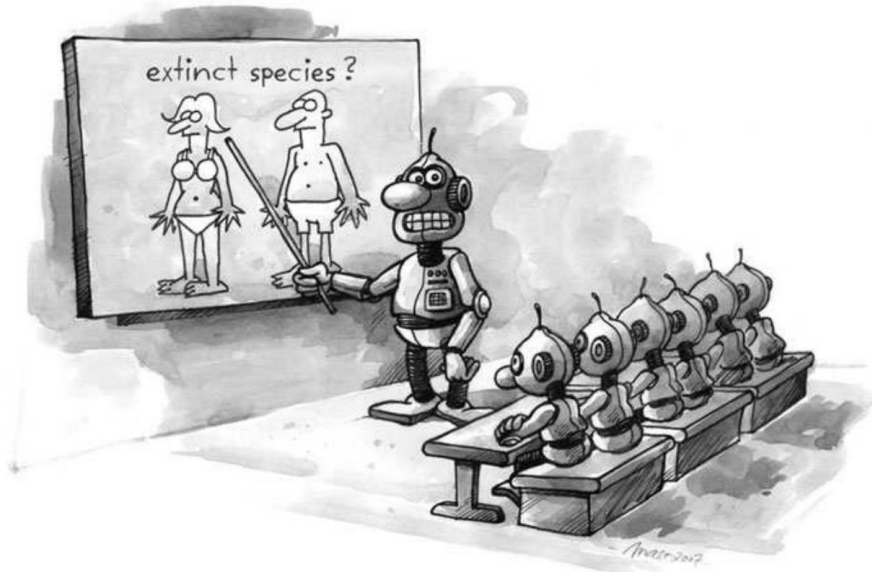


Abb. 8.1 Überwachtes Lernen. . .

Diese Aufgaben sind jedoch recht einfach, da sie kaum mehr als das Speichern von Text in einer Datei umfassen. Daher ist das Auswendiglernen für die KI uninteressant. Im Gegensatz dazu erfolgt der Erwerb mathematischer Fähigkeiten meist nicht durch Auswendiglernen. Bei der Addition natürlicher Zahlen ist dies überhaupt nicht möglich, da es für jeden Term in der Summe $x + y$ unendlich viele Werte gibt. Für jede Kombination der beiden Werte x und y müsste das Tripel $(x, y, x + y)$ gespeichert werden, was unmöglich ist. Bei Dezimalzahlen ist dies völlig unmöglich. Dabei stellt sich die Frage: Wie lernen wir Mathematik? Die Antwort lautet: Der Lehrer erklärt den Prozess und die Schüler üben ihn an Beispielen, bis sie an neuen Beispielen keine Fehler mehr machen. Nach 50 Beispielen versteht der Schüler (hoffentlich) die Addition. Das heißt, nach nur 50 Beispielen kann er das Gelernte auf unendlich viele neue Beispiele anwenden, die bis zu diesem Zeitpunkt noch nicht gesehen wurden. Dieser Vorgang wird als *Generalisierung bezeichnet*. Wir beginnen mit einem einfachen Beispiel.

Beispiel 8.1 Ein Obstbauer möchte geerntete Äpfel automatisch in die Warenklassen A und B einteilen. Die Sortieranlage ist mit Sensoren ausgestattet, um zwei *Merkmale*, Größe und Farbe, zu messen und dann zu entscheiden, zu welcher der beiden Klassen der Apfel gehört. Dies ist eine typische *Klassifizierungsaufgabe*. Systeme, die Merkmalsvektoren in eine endliche Anzahl von Klassen unterteilen können, werden als *Klassifikatoren bezeichnet*.

Zur Konfiguration der Maschine werden die Äpfel von einem Fachmann handverlesen, also klassifiziert. Anschließend werden die beiden Messungen zusammen mit ihrer Klassenbezeichnung in eine Tabelle eingetragen (Tabelle 8.1 auf Seite 163). Die Größe wird in Form des Durchmessers in Zentimetern und die Farbe durch einen Zahlenwert zwischen 0 (für Grün) und

Tabelle 8.1 Trainingsdaten für den Apfelsortieragenten

Größe [cm]	8863. . .
Farbe	0,1 0,3 0,9 0,8 . . .
Warenklasse BAAB	. . .

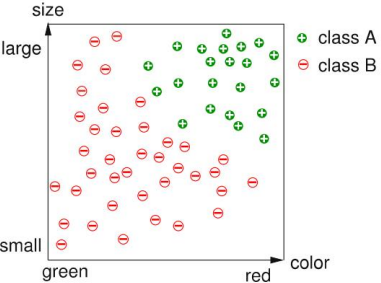
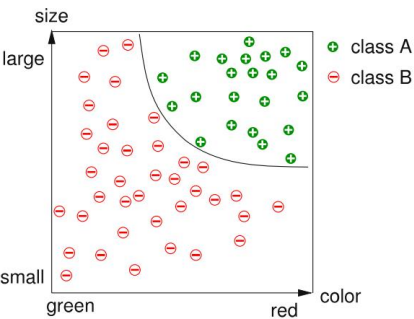


Abb. 8.2 Apfelsortieranlage der BayWa-Firma in Kressbronn und Einordnung einiger Äpfel in die Warenklassen A und B im Objektraum (Foto: BayWa)

Abb. 8.3 Die in das Diagramm eingezeichnete *Kurve* unterteilt die Klassen und kann dann auf beliebige neue Äpfel angewendet werden



Eine Visualisierung der Daten ist als Punkte in einem Streudiagramm rechts in [Abb. 8.2](#) aufgeführt.

Die Aufgabe beim maschinellen Lernen besteht darin, aus den gesammelten, klassifizierten Daten eine Funktion zu generieren, die aus den beiden Merkmalen Größe und Farbe den Klassenwert (A oder B) für einen neuen Apfel berechnet. In [Abb. 8.3](#) ist eine solche Funktion durch die durch das Diagramm gezogene Trennlinie dargestellt. Alle Äpfel mit einem Merkmalsvektor unten links in der Linie werden in die Klasse B eingeordnet, alle anderen in die Klasse A.

In diesem Beispiel ist es noch sehr einfach, eine solche Trennlinie für die beiden Klassen zu finden. Es ist eindeutig eine schwierigere und vor allem viel weniger visualisierbare Aufgabe, wenn die zu klassifizierenden Objekte nicht nur durch zwei, sondern durch viele Merkmale beschrieben werden. In der Praxis werden üblicherweise 30 oder mehr Features verwendet. Für n Merkmale besteht die Aufgabe darin, eine n -dimensionale Hyperebene innerhalb des n -dimensionalen *Merkmalsraums* zu finden, die die Klassen möglichst gut unterteilt. Eine „gute“ Einteilung bedeutet, dass der Anteil falsch klassifizierter Objekte möglichst gering ist.

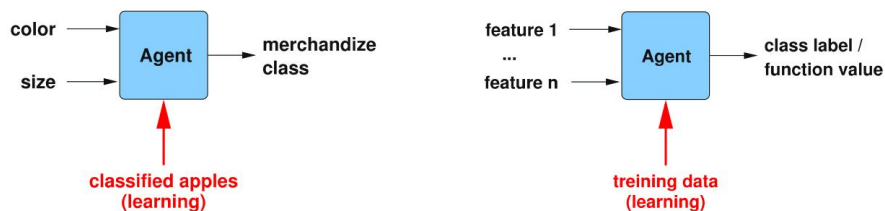


Abb. 8.4 Funktionsstruktur eines Lernagenten zur Apfelsortierung (*links*) und allgemein (*rechts*)

Ein Klassifikator ordnet einen Merkmalsvektor einem Klassenwert zu. Hier gibt es eine feste, meist kleine Anzahl an Alternativen. Die gewünschte Abbildung wird auch **Zielfunktion** genannt. Wenn sich die Zielfunktion nicht auf einen endlichen Bereich abbilden lässt, handelt es sich nicht um eine Klassifikation, sondern um ein **Näherungsproblem**. Ein solches Näherungsproblem ist die Bestimmung des Marktwerts einer Aktie aus gegebenen Merkmalen. In den folgenden Abschnitten stellen wir mehrere Lernagenten für beide Arten von Zuordnungen vor.

Der Lernagent Wir können einen Lernagenten formal als eine Funktion beschreiben, die einen Merkmalsvektor auf einen diskreten Klassenwert oder allgemein auf eine reelle Zahl abbildet. Diese Funktion ist nicht programmiert, sondern entsteht bzw. verändert sich während der *Lernphase*, beeinflusst durch die *Trainingsdaten*. In Abb. 8.4 ist ein solcher Agent am Beispiel der Apfelsortierung dargestellt. Beim Lernen wird der Agent mit den bereits klassifizierten Daten aus Tabelle 8.1 auf Seite 163 gefüttert. Anschließend stellt der Agent eine möglichst gute Abbildung vom Merkmalsvektor auf den Funktionswert (z. B. Warenklasse) her.

Wir können nun versuchen, uns einer Definition des Begriffs „maschinelles Lernen“ anzunähern. Tom Mitchell [Mit97] gibt diese Definition:
 „Maschinelles Lernen ist das Studium von Computeralgorithmen, die Automatisierung verbessern.“
 durch Erfahrung.

Darauf aufbauend geben wir

Definition 8.1 Ein Agent ist ein lernender Agent, wenn er seine Leistung (gemessen an einem geeigneten Kriterium) bei neuen, unbekannten Daten im Laufe der Zeit verbessert (nachdem er viele Trainingsbeispiele gesehen hat).

Es ist wichtig, die Generalisierungsfähigkeit des Lernalgorithmus an unbekannten Daten, den *Testdaten*, zu *testen*. Andernfalls würde jedes System, das gerade die Trainingsdaten gespeichert hat, allein durch den Aufruf der gespeicherten Daten eine optimale Leistung erbringen. Ein lernender Agent wird durch folgende

Begriffe charakterisiert: **Aufgabe:** Die Aufgabe des lernenden Algorithmus besteht darin, eine Abbildung zu lernen. Dies könnte beispielsweise die Zuordnung der Größe und Farbe eines Apfels zu seiner Handelsklasse sein, aber auch die Zuordnung der 15 Symptome eines Patienten zur Entscheidung, ob sein Blinddarm entfernt werden soll oder nicht.



Abb. 8.5 Data Mining

Variabler Agent (genauer gesagt eine Klasse von Agenten): Hier müssen wir entscheiden, mit welchem Lernalgorithmus gearbeitet wird. Wenn dies gewählt wurde, wird somit die Klasse aller lernbaren Funktionen bestimmt.

Trainingsdaten (Erfahrung): Die Trainingsdaten (Probe) enthalten das Wissen, das der Lernalgorithmus extrahieren und lernen soll. Bei der Auswahl der Trainingsdaten muss darauf geachtet werden, dass es sich um eine repräsentative Stichprobe für die zu erlernende Aufgabe handelt.

Testdaten: wichtig für die Beurteilung, ob der trainierte Agent die Trainingsdaten gut auf neue Daten verallgemeinern kann.

Leistungsmaß: Bei der Apfelsortieranlage die Anzahl der korrekt klassifizierten Äpfel. Wir benötigen es, um die Qualität eines Agenten zu testen. Die Kenntnis des Leistungsmaßes ist in der Regel viel einfacher als die Kenntnis der Funktion des Agenten. Beispielsweise ist es einfach, die Leistung (Zeit) eines 10.000-Meter-Läufers zu messen. Dies bedeutet jedoch keineswegs, dass der Schiedsrichter, der die Zeit misst, genauso schnell laufen kann. Der Schiedsrichter weiß nur, wie er die Leistung messen soll, nicht aber die „Funktion“ des Agenten, dessen Leistung er misst.

Was ist Data Mining? Die Aufgabe einer lernenden Maschine ist es, aus Trainingsdaten Wissen zu extrahieren. Oftmals möchte der Entwickler oder Anwender, dass die lernende Maschine das extrahierte Wissen auch für Menschen lesbar macht. Noch besser ist es, wenn der Entwickler das Wissen sogar ändern kann. Der Prozess der Induktion von Entscheidungsbäumen in Abschn. 8.4 ist ein Beispiel für eine solche Methode.

Ähnliche Herausforderungen ergeben sich aus dem elektronischen Geschäftsverkehr und dem Wissensmanagement. Hier zeigt sich ein klassisches Problem: Von den Aktionen der Besucher auf seinem Webportal,

Der Inhaber eines Internetunternehmens möchte eine Beziehung zwischen dem herstellen Eigenschaften eines Kunden und die Produktklasse, die für ihn interessant ist Kunde. Dann kann ein Verkäufer kundenspezifische Anzeigen schalten. Das Dies wird anschaulich auf www.amazon.com demonstriert, wo dem Kunden Produkte empfohlen werden, die denen des vorherigen Besuchs ähneln. In vielen In den Bereichen Werbung und Marketing sowie im Customer Relationship Management (CRM) kommen Data-Mining-Techniken zum Einsatz. Immer dann, wenn es um große Mengen geht Je mehr Daten vorhanden sind, desto mehr kann man versuchen, diese Daten für die Kundenanalyse zu nutzen Präferenzen, um kundenspezifische Werbung anzuzeigen. Das aufstrebende Feld Diesem Zweck ist die Studie des Präferenzlernens gewidmet.

Der Prozess der Wissensgewinnung aus Daten sowie deren Darstellung und Anwendung wird als *Data Mining* bezeichnet. Die verwendeten Methoden werden in der Regel übernommen aus Statistik oder maschinellem Lernen und sollte auf sehr große Bereiche anwendbar sein Datenmengen zu angemessenen Kosten.

Im Rahmen der Informationsbeschaffung, beispielsweise im Internet oder in einem Intranet, spielt Text Mining eine immer wichtigere Rolle. Typische Aufgaben sind das Auffinden ähnlicher Texte in einer Suchmaschine oder das Klassifizieren von Texten, die z.B wird in Spamfiltern für E-Mails angewendet. In Abschn. 8.6.1 werden wir das weit verbreitete vorstellen Naiver Bayes-Algorithmus zur Klassifizierung von Text. Eine relativ neue Herausforderung für Unter Data Mining versteht man die Extraktion struktureller, statischer und dynamischer Informationen aus Diagrammstrukturen wie soziale Netzwerke, Verkehrsnetzwerke oder Internetverkehr.

Da sich die beiden beschriebenen Aufgaben des maschinellen Lernens und des Data Mining weitgehend ähneln, sind die grundlegenden Methoden, die in beiden Bereichen zum Einsatz kommen, größtenteils identisch. Daher wird in der Beschreibung der Lernalgorithmen nicht unterschieden zwischen maschinellem Lernen und Data Mining.

Aufgrund der enormen kommerziellen Auswirkungen von Data-Mining-Techniken gibt es diese mittlerweile viele ausgefeilte Optimierungen und eine ganze Reihe leistungsstarker Data-Mining-Systeme, die eine große Palette praktischer Werkzeuge zur Wissensextraktion bieten Daten. Ein solches System wird in Abschn. 2.1 vorgestellt. 8.8.

8.1 Datenanalyse

Statistiken bieten eine Reihe von Möglichkeiten, Daten mit einfachen Parametern zu beschreiben. Aus Aus diesen wählen wir einige aus, die für die Trainingsanalyse besonders wichtig sind Daten und testen Sie diese an einer Teilmenge der LEXMED- Daten aus Abschn. 7.3. In diesem Beispiel Datensatz, die Symptome $x_1, \dots, x_{15}$ von $N = 473$ Patienten, prägnant beschrieben in Tabelle 8.2 auf Seite 167, sowie die Klassenbezeichnung – also die Diagnose (Blinddarmentzündung). positiv/negativ) – werden aufgelistet. Patient Nummer eins wird zum Beispiel von beschrieben Vektor

$$x_1 = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37,9, 38,8, 23100, 0, 1)$$

Tabelle 8.2 Beschreibung von Variablen $x_1, \dots, x_{16}$.

Ein etwas anderes
Formalisierung wurde in verwendet
Tabelle 7.2 auf Seite 133

Var. Num.	Beschreibung	Werte
1	Alter	Kontinuierlich
2	Geschlecht (1=männlich, 2=weiblich)	1, 2
3	Schmerzquadrant 1	0, 1
4	Schmerzquadrant 2	0, 1
5	Schmerzquadrant 3	0, 1
6	Schmerzquadrant 4	0, 1
7	Lokaler Muskelschutz	0, 1
8	Generalisierte Muskelprotektion	0, 1
9	Rebound-Zärtlichkeit	0, 1
10	Schmerzen beim Klopfen	0, 1
11	Schmerzen bei rektaler Untersuchung	0, 1
12	Axiale Temperatur	Kontinuierlich
13	Rektale Temperatur	Kontinuierlich
14	Leukozyten	Kontinuierlich
15	Diabetes Mellitus	0, 1
16	Appendizitis	0, 1

und Patient Nummer zwei von

$$2x_{-} = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36,9, 37,4, 8100, 0, 0)$$

Patient Nummer zwei hat den Leukozytenwert $x_{14}^2 = 8100$.

Für jede Variable x_i ist ihr Durchschnitt $\bar{x}_i$ definiert als

$$\bar{x}_i := \frac{1}{N} \sum_{p=1}^N x_{ip}$$

und die Standardabweichung s_i als Maß für seine durchschnittliche Abweichung vom Durchschnitt Wert als

$$s_i := \sqrt{\frac{1}{N-1} \sum_{p=1}^N (x_{ip} - \bar{x}_i)^2}$$

Für die Analyse mehrdimensionaler Daten ist die Frage wichtig, ob zwei Variablen x_i und x_j statistisch abhängig (korreliert) sind. Zum Beispiel die Kovarianz

$$\bar{y}_{ij} = \frac{1}{N-1} \sum_{p=1}^N (x_{ip} - \bar{x}_i)(x_{jp} - \bar{x}_j)$$

Tabelle 8.3 Korrelationsmatrix für die 16 in 473 Fällen gemessenen Blinddarmentzündungsvariablen

1.	y0,009 0,14	0,037 y0,096 0,12		0,018 0,051 y0,034 y0,041 0,034 0,037 0,05 y0,037 0,37										0,012		
y0,009 1.	y0,0074 y0,019 y0,06 0,063 y0,17			0,0084 y0,17 y0,14 y0,13 y0,017 y0,034 y0,14										0,045 y0,2		
0,14 y0,0074 1.	0,55	y0,091 0,24		0,13	0,24	0,045 0,18		0,028 0,02		0,045 0,03		0,11	0,045			
0,037 y0,019 0,55	1.	y0,24 0,33		0,051 0,25		0,074 0,19		0,087 0,11		0,12	0,11	0,14 y0,0091				
y0,096 y0,06 y0,091 y0,24				1.		0,059 0,14		0,034		0,14	0,049 0,057 0,064 0,058 0,11					
												0,017		0,14		
0,12	0,063	0,24	0,33	0,059 1.		0,071 0,19		0,086 0,15		0,048 0,11		0,12	0,063	0,21	0,053	
0,018 y0,17		0,13	0,051	0,14 0,071 1.		0,16		0,4	0,28	0,2	0,24	0,36	0,29 y0,0001 0,33			
0,051 0,0084 0,24			0,25		0,034 0,19		0,16	1.	0,17	0,23	0,24	0,19	0,24	0,27	0,083	0,084
y0,034 y0,17		0,045	0,074	0,14 0,086 0,4		0,17	1.	0,53	0,25	0,19	0,27	0,27	0,026 0,38			
y0,041 y0,14		0,18	0,19	0,049 0,15		0,28	0,23	0,53	1.	0,24	0,15	0,19	0,23	0,02	0,32	
0,034 y0,13		0,028	0,087	0,057 0,048 0,2		0,24	0,25	0,24	1.	0,17	0,17	0,22	0,098	0,17		
0,037 y0,017 0,02			0,11	0,064 0,11		0,24	0,19	0,19	0,15	0,17	1.	0,72	0,26	0,035	0,15	
0,05 y0,034 0,045			0,12	0,058 0,12		0,36	0,24	0,27	0,19	0,17	0,72	1.	0,38	0,044	0,21	
y0,037 y0,14		0,03	0,11	0,11 0,063 0,29		0,27	0,27	0,23	0,22	0,26	0,38	1.	0,051	0,44		
0,37	0,045	0,11	0,14	0,017 0,21 y0,0001 0,083				0,026 0,02		0,098 0,035 0,044 0,051				1.	y0,0055	
0,012 y0,2		0,045 y0,0091 0,14 0,053 0,33				0,084		0,38	0,32	0,17	0,15	0,21	0,44 y0,0055 1.			

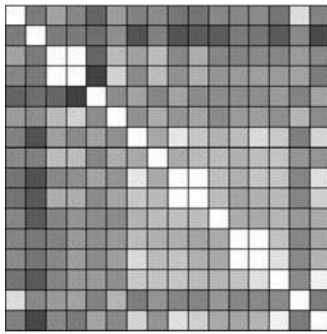
gibt Auskunft darüber. In dieser Summe liefert der Summand einen positiven Eintrag für der p-te Datenvektor genau dann, wenn die Abweichungen der i-ten und j-ten Komponenten von im Durchschnitt haben beide das gleiche Vorzeichen. Wenn sie unterschiedliche Vorzeichen haben, dann ist der Eintrag Negativ. Daher sollte die Kovarianz $\bar{y}_{12,13}$ der beiden unterschiedlichen Fieberwerte betragen ganz klar positiv.

Allerdings hängt die Kovarianz auch vom Absolutwert der Variablen ab, was den Vergleich der Werte erschwert. Um den Abschluss vergleichen zu können Als Abhängigkeit bei mehreren Variablen definieren wir daher die *Korrelation Koeffizient*

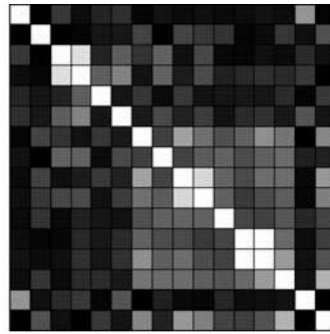
$$K_{ij} = \frac{\bar{y}_{ij}}{s_i \cdot s_j}$$

für zwei Werte x_i und x_j , was nichts anderes als eine normalisierte Kovarianz ist. Die Matrix K aller Korrelationskoeffizienten enthält Werte zwischen $\bar{y}_1$ und 1, ist symmetrisch, und alle seine Diagonalelemente haben den Wert 1. Die Korrelationsmatrix für alle 16 Variablen sind in [Tabelle 8.3 angegeben](#).

Diese Matrix wird etwas besser lesbar, wenn wir sie als Dichte darstellen Handlung. Anstelle der Zahlenwerte die Matricelemente in [Abb. 8.6](#) auf Seite 169 sind mit Grauwerten gefüllt. Im rechten Diagramm sind die absoluten Werte dargestellt. Daher Wir können sehr schnell erkennen, welche Variablen eine schwache oder starke Abhängigkeit aufweisen. Wir Man erkennt zum Beispiel, dass die Variablen 7, 9, 10 und 14 die stärkste Korrelation mit der Klassenvariable *Blinddarmentzündung* aufweisen und daher für die wichtiger sind Diagnose als die andere Variable. Wir sehen jedoch auch, dass die Variablen 9 und 10 sind stark korreliert. Dies könnte bedeuten, dass einer dieser beiden Werte potenziell vorhanden ist ausreichend für die Diagnose.



$K_{ij} = -1$: black, $K_{ij} = 1$: white

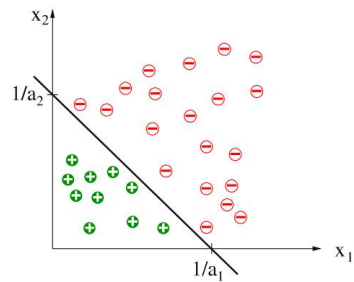


$|K_{ij}| = 0$: black, $|K_{ij}| = 1$: white

Abb. 8.6 Die Korrelationsmatrix als Häufigkeitsgraph. Im *linken Diagramm* steht *Dunkel* für Negativ und *Hell* für Positiv. Im *rechten Bild* wurden die absoluten Werte aufgeführt. Dabei bedeutet *Schwarz* $K_{ij} \approx 0$ (unkorreliert) und *Weiß* $|K_{ij}| \approx 1$ (stark korreliert)

Abb. 8.7 Ein linear trennbarer zweidimensionaler Datensatz.

Die Gleichung für die Teilungsgerade lautet $a_1x_1 + a_2x_2 = 1$



8.2 Das Perzeptron, ein linearer Klassifikator

Im Beispiel der Sortierung von Äpfeln ist in Abb. 8.3 auf Seite 163 eine gekrümmte Trennlinie zwischen den beiden Klassen eingezeichnet. Ein einfacherer Fall ist in Abb. 8.7 dargestellt. Hier können die zweidimensionalen Trainingsbeispiele durch eine Gerade getrennt werden. Wir nennen einen solchen Satz von Trainingsdaten *linear trennbar*. In n Dimensionen wird zur Trennung eine Hyperebene benötigt. Dies stellt einen linearen Unterraum der Dimension $n - 1$ dar.

Weil jede $(n - 1)$ -dimensionale Hyperebene in der $\mathbb{R}^n$ kann durch ein beschrieben werden Gleichung

$$\sum_{i=1}^n a_i x_i = \tilde{y}$$

Es ist sinnvoll, die lineare Trennbarkeit wie folgt zu definieren.

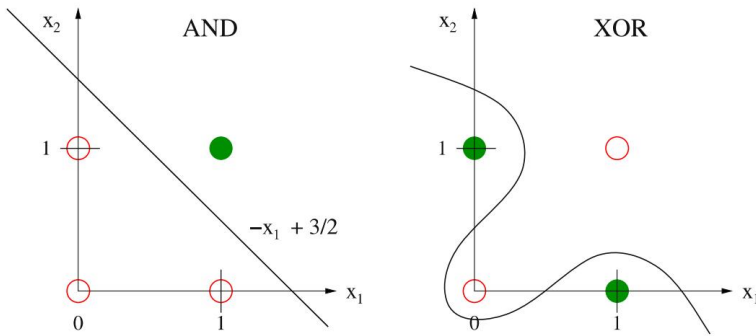


Abb. 8.8 Die boolesche Funktion AND ist linear separierbar, XOR jedoch nicht ($\bullet$ = wahr, $\circ$ = falsch)

Definition 8.2 Zwei Mengen $M_1 \subseteq \mathbb{R}^N$ und $M_2 \subseteq \mathbb{R}^N$ heißen *linear separierbar* R , wenn reelle Zahlen $a_1, \dots, a_N, \gamma$ existieren mit

$$a_i x_i > \gamma \text{ für alle } x \in M_1 \text{ und}$$

$$a_i x_i \leq \gamma \text{ für alle } x \in M_2.$$

Der Wert γ wird als Schwelle bezeichnet.

In Abb. 8.8 sehen wir, dass die AND-Funktion linear separierbar ist, die XOR-Funktion jedoch nicht. Für AND trennt beispielsweise die Linie $\gamma x_1 + 3/2$ wahre und falsche Interpretationen der Formel $x_1 \wedge x_2$. Im Gegensatz dazu hat die XOR-Funktion keine gerade Trennlinie. Offensichtlich hat die XOR-Funktion in dieser Hinsicht eine komplexere Struktur als die AND-Funktion.

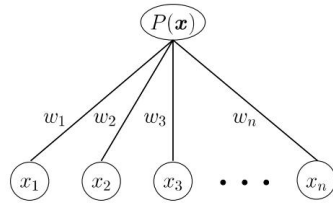
Mit dem Perzeptron stellen wir einen sehr einfachen Lernalgorithmus vor, der separieren linear trennbare Mengen.

Definition 8.3 Sei $w = (w_1, \dots, w_N) \in \mathbb{R}^N$ ein Gewichtsvektor und $x \in \mathbb{R}^N$ Eingabevektor. Ein *Perzeptron* stellt eine Funktion P dar: $\mathbb{R}^N \rightarrow \{0, 1\}$ was entspricht der folgenden Regel:

$$P(x) = \begin{cases} 1 & \text{wenn } wx = \sum_{i=1}^N w_i x_i > 0, \\ 0 & \text{sonst.} \end{cases}$$

Das Perzeptron [Ros58, MP69] ist ein sehr einfacher Klassifizierungsalgorithmus. Es entspricht einem zweischichtigen neuronalen Netzwerk mit Aktivierung durch eine Schwellenwertfunktion, dargestellt in Abb. 8.9 auf Seite 171. Wie in Kap. In 9 repräsentiert jeder Knoten im Netzwerk ein Neuron und jede Kante eine Synapse. Im Moment werden wir jedoch nur

Abb. 8.9 Grafische Darstellung eines Perzeptrons als zweischichtiges neuronales Netzwerk



Betrachten Sie das Perzeptron als Lernagent, das heißt als mathematische Funktion, die einen Merkmalsvektor auf einen Funktionswert abbildet. Dabei werden die Eingabevariablen x_i als Features bezeichnet.

Wie wir in der Formel $\sum_{i=1}^N w_i x_i > 0$ sehen können, werden alle Punkte x oberhalb der Hyperebene $\sum_{i=1}^N w_i x_i = 0$ als positiv ($P(x) = 1$) und alle anderen als negativ ($P(x) = 0$) klassifiziert. Die trennende Hyperebene geht durch den Ursprung, weil $y = 0$.

Mit einem kleinen Trick zeigen wir, dass das Fehlen einer willkürlichen Schwelle keine Einschränkung der Macht darstellt. Zunächst wollen wir jedoch einen einfachen Lernalgorithmus für das Perzeptron vorstellen.

8.2.1 Die Lernregel

Mit der Notation M^+ und M^- für die Sätze positiver bzw. negativer Trainingsmuster lautet die Perzeptron-Lernregel [MP69]

```

PERCEPTRONLEARNING[ $M^+, M^-$ ]  $w =$ 
beliebiger Vektor reeller Zahlen
Wiederholen
  Für alle  $x \in M^+$ 
    Wenn  $w \cdot x \leq 0$ , dann ist  $w = w + x$ 
  Für alle  $x \in M^-$ 
    Wenn  $w \cdot x > 0$ , dann ist  $w = w - x$ 
Bis alle  $x \in M^+ \cup M^-$  korrekt klassifiziert sind
  
```

Das Perzeptron sollte für alle $x \in M^+$ den Wert 1 ausgeben. Nach Definition 8.3 auf Seite 170 ist dies der Fall, wenn $w \cdot x > 0$. Ist dies nicht der Fall, wird x zum Gewichtsvektor w addiert, wodurch der Gewichtsvektor genau in die richtige Richtung geändert wird. Wir sehen dies, wenn wir das Perzeptron auf den geänderten Vektor $w + x$ anwenden, weil

$$2(w + x) \cdot x = w \cdot x + x \cdot x > w \cdot x$$

Wenn dieser Schritt oft genug wiederholt wird, wird der Wert $w \cdot x$ irgendwann positiv, wie es sein sollte. Analog sehen wir, dass für negative Trainingsdaten die

perceptron errechnet einen immer kleineren Wert

$$(w - \tilde{y} x) \cdot x = wx - \tilde{y} x^2$$

was irgendwann negativ wird.²

Beispiel 8.2 Ein Perzeptron soll auf den Mengen $M_+ = \{(0, 1, 8), (2, 0, 6)\}$ und $M_- = \{(\tilde{y}1, 2, 1, 4), (0, 4, \tilde{y}1)\}$ trainiert werden. $w = (1, 1)$ wurde als anfänglicher Gewichtsvektor verwendet. Die Trainingsdaten und die durch den Gewichtsvektor $wx = x_1 + x_2 = 0$ definierte Linie sind in Abb. 8.10 auf Seite 173 im ersten Bild in der oberen Reihe dargestellt. Zusätzlich wird der Gewichtsvektor als gestrichelte Linie eingezeichnet. Da $wx = 0$ ist, ist dies orthogonal zur Gerade

In der ersten Iteration durch die Schleife des Lernalgorithmus ist das nur falsch. Das klassifizierte Trainingsbeispiel ist $(\tilde{y}1, 2, 1, 4)$, weil

$$(\tilde{y}1, 2, 1, 4) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0,2 > 0.$$

Daraus ergibt sich $w = (1, 1) - (\tilde{y}1, 2, 1, 4) = (2, 2, \tilde{y}0, 4)$, wie im zweiten Bild in der oberen Reihe in Abb. 8.10 auf Seite 173 gezeichnet. Die anderen Bilder zeigen, wie es danach geht insgesamt fünf Änderungen, die Trennlinie verläuft zwischen den beiden Klassen. Das Perzeptron klassifiziert somit alle Daten korrekt. Wir sehen im Beispiel deutlich, dass jeder falsch klassifizierte Datenpunkt aus M_+ den Gewichtsvektor w in seine Richtung „zieht“ und jeder falsch klassifizierte Punkt aus M_- den Gewichtsvektor in die entgegengesetzte Richtung „drückt“.

Es wurde gezeigt [MP69], dass das Perzeptron für lineare Separierungen immer konvergiert. Ackerdaten. Wir haben

Satz 8.1 Seien die Klassen M_+ und M_- durch eine Hyperebene $wx = 0$ linear trennbar. Dann konvergiert PERCEPTRONLEARNING für jede Initialisierung des Vektors w . Das Perzeptron P mit dem so berechneten Gewichtsvektor teilt die Klassen M_+ und M_- , das heißt:

$$P(x) = 1 \text{ } \tilde{y} \text{ } x \in M_+$$

Und

$$P(x) = 0 \text{ } \tilde{y} \text{ } x \in M_-.$$

Wie wir in Beispiel 8.2 deutlich sehen können, können Perzeptrene wie oben definiert keine beliebigen linear trennbaren Mengen teilen, sondern nur solche, die durch eine Linie durch den Ursprung oder in $\mathbb{R}^N$ durch eine Hyperebene durch den Ursprung teilbar sind, da der konstante Term $\tilde{y}$ ist fehlt in der Gleichung $\sum_{i=1}^N w_i x_i = 0$.

²Vorsicht! Dies ist kein Konvergenzbeweis für die Perzeptron-Lernregel. Es zeigt nur, dass das Perzeptron konvergiert, wenn der Trainingsdatensatz aus einem einzelnen Beispiel besteht.

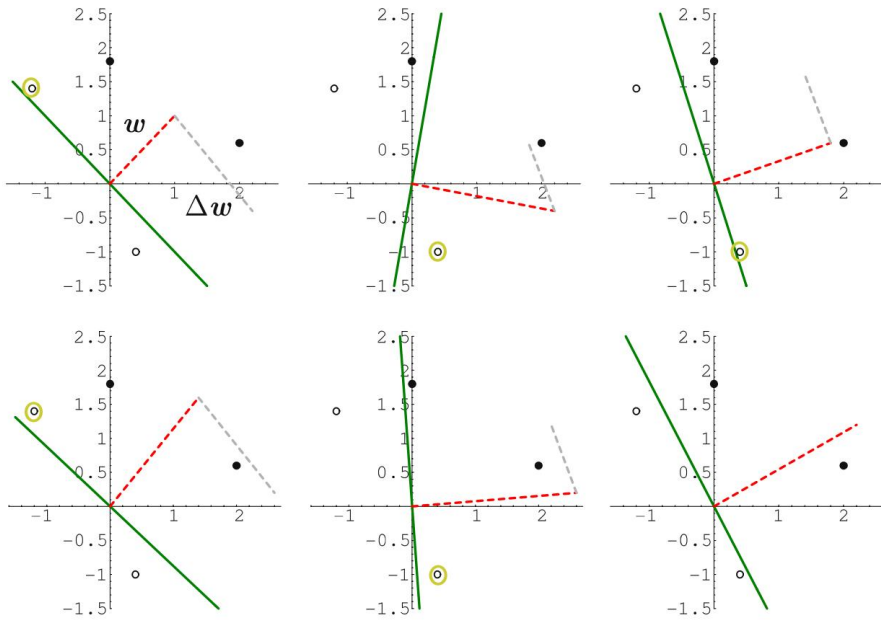


Abb. 8.10 Anwendung der Perzeptron-Lernregel auf zwei positive (-) und zwei negative (•) Datenpunkte. Die *durchgezogene Linie* zeigt die aktuelle Trennlinie $wx = 0$. Die *orthogonale gestrichelte Linie* ist der Gewichtsvektor w und die *zweite gestrichelte Linie* der hinzuzufügende Änderungsvektor $w = x$ oder $w = \tilde{y}x$, der aus dem aktuell aktiven Datenpunkt berechnet wird grau umrandet

Mit dem folgenden Trick können wir den konstanten Term erzeugen. Wir halten die letzte Komponente x_n des Eingabevektors x konstant und setzen sie auf den Wert 1. Nun funktioniert das Gewicht $w_n =: \tilde{y}$ wie ein Schwellenwert, weil

$$\begin{array}{ccc} \sum_{i=1}^n w_i x_i = & \sum_{i=1}^{n-1} w_i \tilde{y}_i > 0 \tilde{y} & \sum_{i=1}^{n-1} w_i x_i > \tilde{y}. \end{array}$$

Ein solcher konstanter Wert $x_n = 1$ in der Eingabe wird als *Bias-Einheit bezeichnet*. Da das damit verbundene Gewicht eine ständige Verschiebung der Hyperebene bewirkt, passt der Begriff „Bias“ gut.

Bei der Anwendung des Perzeptron-Lernalgorithmus wird an den Trainingsdatenvektor ein Bit mit dem konstanten Wert 1 angehängt. Wir beobachten, dass das Gewicht w_n oder der Schwellenwert $\tilde{y}$ während des Lernprozesses gelernt.

Nun wurde gezeigt, dass ein Perzeptron $\tilde{P} : \mathbb{R}^{n-1} \rightarrow \{0, 1\}$

$$\tilde{P}(x_1, \dots, x_{n-1}) = \begin{cases} 1 & \text{wenn } \sum_{i=1}^{n-1} w_i x_i > \tilde{y}, \\ 0 & \text{sonst} \end{cases} \quad (8.1)$$

Mit einem beliebigen Schwellenwert kann durch ein Perzeptron $P : \mathbb{R}^n \rightarrow \{0, 1\}$ mit Schwellenwert 0 simuliert werden. Vergleichen wir (8.1) mit der Definition von linear trennbar, dann sehen wir, dass beide Aussagen äquivalent sind. Zusammenfassend haben wir Folgendes gezeigt:

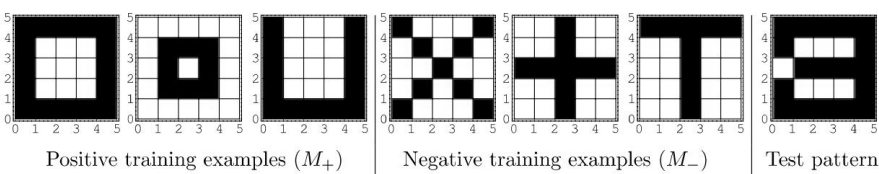
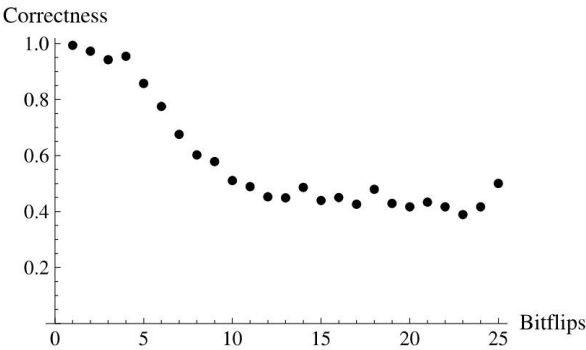


Abb. 8.11 Die sechs für das Training verwendeten Muster. Das *gesamte rechte Muster* ist eines der 22 Testmuster für das *erste Muster* mit einer Folge von vier invertierten Bits

Abb. 8.12 Relative Korrektheit des Perzeptrons als Funktion der Anzahl invertierter Bits in den Testdaten



Satz 8.2 Eine Funktion $f : \mathbb{R}^n \rightarrow \{0, 1\}$ kann genau dann durch ein Perzeptron dargestellt werden, wenn die beiden Mengen positiver und negativer Eingabevektoren linear trennbar sind.

Beispiel 8.3 Wir trainieren nun ein Perzeptron mit einem Schwellenwert auf sechs einfachen, grafischen Binärmustern, dargestellt in Abb. 8.11, mit jeweils 5×5 Pixeln.

Die Trainingsdaten können von PERCEPTRONLEARNING in vier Iterationen über alle Muster gelernt werden. Muster mit einer variablen Anzahl invertierter Bits, die als Rauschen eingeführt werden, werden verwendet, um die Generalisierungsfähigkeit des Systems zu testen. Die invertierten Bits im Testmuster liegen jeweils der Reihe nach hintereinander vor. In Abb. 8.12 ist der Prozentsatz korrekt klassifizierter Muster als Funktion der Anzahl falscher Bits aufgetragen.

Nach etwa fünf aufeinanderfolgenden invertierten Bits nimmt die Korrektheit stark ab, was angesichts der Einfachheit des Modells nicht verwunderlich ist. Im nächsten Abschnitt stellen wir einen Algorithmus vor, der in diesem Fall deutlich besser abschneidet.

8.2.2 Optimierung und Ausblick

Als einer der einfachsten auf neuronalen Netzwerken basierenden Lernalgorithmen kann das zweischichtige Per-Zeptron nur linear trennbare Klassen unterteilen. In Abschn. 9.5 werden wir sehen, dass mehrschichtige Netzwerke deutlich leistungsfähiger sind. Trotz seiner einfachen Str

Das Perzeptron in der dargestellten Form konvergiert sehr langsam. Es kann durch Normalisierung des gewichtsverändernden Vektors beschleunigt werden. Die Formeln $w = w \pm x$ werden durch $w = w \pm x/|x|$ ersetzt. Dadurch hat jeder Datenpunkt beim Lernen das gleiche Gewicht, unabhängig von seinem Wert.

Die Konvergenzgeschwindigkeit hängt stark von der Initialisierung des Vektors w ab. Im Idealfall müsste es überhaupt nicht geändert werden und der Algorithmus würde nach einer Iteration konvergieren. Durch den Einsatz der heuristischen Initialisierung können wir diesem Ziel näher kommen

$$w_0 = \frac{\sum_{i=1}^M x_i y_i}{\sum_{i=1}^M x_i^2 + \sum_{i=1}^M y_i^2} X,$$

was wir in Aufgabe 8.5 auf Seite 217 genauer untersuchen werden.

Vergleicht man die Perzeptronformel mit der in Abschn. 7.3.1 sehen wir sofort ihre Äquivalenz. Darüber hinaus ist das Perzeptron als einfachstes neuronales Netzwerkmodell äquivalent zu Naive Bayes, der einfachsten Art von Bayes'schen Netzwerken (siehe Übung 8.16 auf Seite 219). Somit haben offenbar mehrere sehr unterschiedliche Klassifikationsalgorithmen einen gemeinsamen Ursprung.

Im Kap. 9 lernen wir eine Verallgemeinerung des Perzeptrons in Form des Backpropagation-Algorithmus kennen, der nichtlinear trennbare Mengen durch die Verwendung mehrerer Schichten teilen kann und über eine bessere Lernregel verfügt.

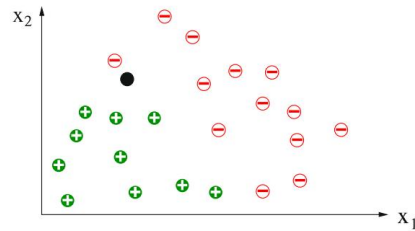
8.3 Die Nearest-Neighbor-Methode

Für ein Perzeptron wird das in den Trainingsdaten vorhandene Wissen extrahiert und in komprimierter Form in den Gewichten w_i gespeichert. Dadurch gehen Informationen über die Daten verloren. Genau das ist jedoch dann erwünscht, wenn das System von den Trainingsdaten auf neue Daten verallgemeinern soll. Die Generalisierung ist in diesem Fall ein zeitintensiver Prozess mit dem Ziel, eine kompakte Darstellung der Daten in Form einer Funktion zu finden, die neue Daten möglichst gut klassifiziert.

Das Speichern aller Daten durch einfaches Speichern ist etwas ganz anderes. Hier ist das Lernen äußerst einfach. Allerdings lässt sich das gespeicherte Wissen, wie bereits erwähnt, nicht so einfach auf neue, unbekannte Beispiele übertragen. Ein solcher Ansatz ist beispielsweise für das Erlernen des Skifahrens sehr ungeeignet. Ein Anfänger kann nie ein guter Skifahrer werden, indem er sich nur Videos von guten Skifahrern ansieht. Offensichtlich passiert bei der automatischen Ausführung solcher Lernbewegungen etwas Ähnliches wie beim Perzeptron. Nach ausreichend langer Übung wird das in Trainingsbeispielen gespeicherte Wissen in eine interne Repräsentation im Gehirn umgewandelt.

Es gibt jedoch gelungene Beispiele des Auswendiglernens, bei denen auch eine Verallgemeinerung möglich ist. Während der Diagnose eines schwierigen Falles könnte ein Arzt versuchen, sich an ähnliche Fälle aus der Vergangenheit zu erinnern. Wenn sein Gedächtnis in Ordnung ist, könnte er auf diesen Fall stoßen, ihn in seinen Akten nachschlagen und schließlich zu einer ähnlichen Diagnose kommen. Für dieses Vorgehen muss der Arzt ein gutes Gespür für *Ähnlichkeiten haben*, um sich an den ähnlichsten Fall zu erinnern. Wenn er dies gefunden hat, muss er sich fragen, ob es ähnlich genug ist, um dieselbe Diagnose zu rechtfertigen.

Abb. 8.13 In diesem Beispiel mit negativen und positiven Trainingsbeispielen gruppiert die Methode des nächsten Nachbarn den neuen, schwarz markierten Punkt in die negative Klasse



Was bedeutet Ähnlichkeit in dem formalen Kontext, den wir konstruieren? Wir stellen die Trainingsbeispiele wie gewohnt in einem mehrdimensionalen Merkmalsraum dar und definieren: *Je kleiner ihr Abstand im Merkmalsraum, desto ähnlicher sind sich zwei Beispiele.*

Wir wenden diese Definition nun auf das einfache zweidimensionale Beispiel aus Abb. 8.13 an. Hier ist der nächste Nachbar des schwarzen Punktes ein negatives Beispiel. Somit wird es der negativen Klasse zugeordnet.

Der Abstand $d(x, y)$ zwischen zwei Punkten $x \in \mathbb{R}^N$ und $y \in \mathbb{R}^N$ kann zum Beispiel sein gemessen durch den euklidischen Abstand

$$d(x, y) = |x - y| = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}.$$

Da es neben dieser noch viele andere Distanzmetriken gibt, ist es sinnvoll, über Alternativen für eine konkrete Anwendung nachzudenken. In vielen Anwendungen sind bestimmte Funktionen wichtiger als andere. Daher ist es oft sinnvoll, die Features durch Gewichte w_i unterschiedlich zu skalieren. Die Formel lautet dann

$$dw(x, y) = |x - y| = \sqrt{\sum_{i=1}^N w_i (x_i - y_i)^2}.$$

Das folgende einfache Programm zur Klassifizierung des nächsten Nachbarn durchsucht die Trainingsdaten nach dem nächsten Nachbar t zum neuen Beispiel s und klassifiziert dann s genau wie t .

```
NEARESTNEIGHBOR[M+,M-,s] t =
  argminx ∈ M+ {d(s, x)}
  Wenn t ∈ M+, dann Return („+“)
  Else Return („-“)
```

³Die Funktionale argmin und argmax bestimmen, ähnlich wie min und max, das Minimum bzw. Maximum einer Menge oder Funktion. Sie geben jedoch nicht den Wert des Maximums oder Minimums zurück, sondern die Position, also das Argument, in dem das Extremum auftritt.

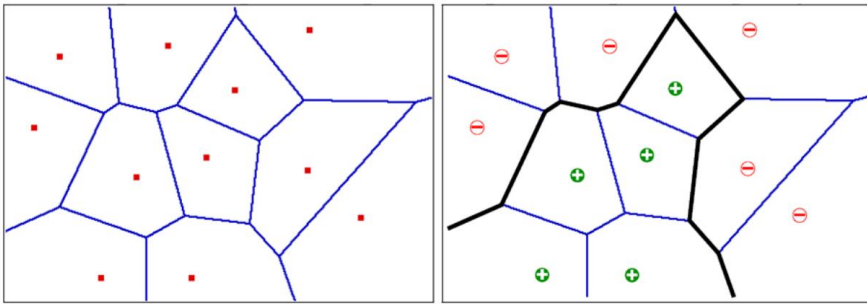
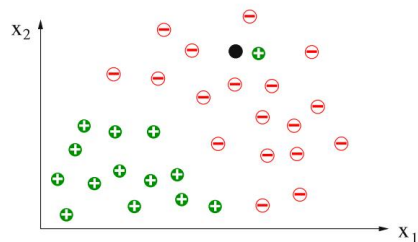


Abb. 8.14 Eine Menge von Punkten zusammen mit ihrem Voronoi-Diagramm (*links*) und der erzeugten *Trennlinie* für die beiden Klassen M_+ und M_-

Abb. 8.15 Die Methode „Nächster Nachbar“ ordnet den neuen, **schwarz** markierten Punkt der falschen (positiven) Klasse zu, da der nächste Nachbar höchstwahrscheinlich falsch klassifiziert ist



Im Gegensatz zum Perzeptron erzeugt die Methode des nächsten Nachbarn keine Linie, die die Trainingsdatenpunkte teilt. Es gibt jedoch sicherlich eine imaginäre Linie, die die beiden Klassen trennt. Wir können dies erzeugen, indem wir zunächst das sogenannte *Voronoi-Diagramm erstellen*. Im Voronoi-Diagramm ist jeder Datenpunkt von einem konvexen Polygon umgeben, das somit eine Umgebung um ihn herum definiert. Das Voronoi-Diagramm hat die Eigenschaft, dass für einen beliebigen neuen Punkt der nächste Nachbar unter den Datenpunkten der Datenpunkt ist, der in derselben Nachbarschaft liegt. Wenn das Voronoi-Diagramm für einen Satz von Trainingsdaten bestimmt wird, ist es einfach, den nächsten Nachbarn für einen neuen zu klassifizierenden Punkt zu finden. Die Klassenzugehörigkeit wird dann vom nächsten Nachbarn übernommen.

In Abb. 8.14 sehen wir deutlich, dass die Methode des nächsten Nachbarn deutlich leistungsfähiger ist als das Perzeptron. Es ist in der Lage, beliebig komplexe Trennlinien (allgemein: Hyperebenen) korrekt darzustellen. Allerdings besteht hier eine Gefahr. Ein einzelner fehlerhafter Punkt kann unter Umständen zu sehr schlechten Klassifizierungsergebnissen führen. Ein solcher Fall tritt in Abb. 8.15 bei der Klassifikation des Schwarzpunktes auf. Die Methode des nächsten Nachbarn klassifiziert dies möglicherweise falsch. Wenn der schwarze Punkt unmittelbar neben einem positiven Punkt liegt, der ein Ausreißer der positiven Klasse ist, wird er als positiv und nicht wie hier beabsichtigt als negativ klassifiziert. Eine fehlerhafte Anpassung an zufällige Fehler (Rauschen) wird als *Überanpassung bezeichnet*.

K-NEARESTNEIGHBOR(M_+, M_-, s)

$V = \{k \text{ nächste Nachbarn in } M_+ \text{ } \bar{y} \text{ } M_-\}$

Wenn $|M_+ \bar{y} V| > |M_- \bar{y} V|$ Dann Return(„+“)

Elseif $|M_+ \bar{y} V| < |M_- \bar{y} V|$ Dann Return(„-“)

Anders

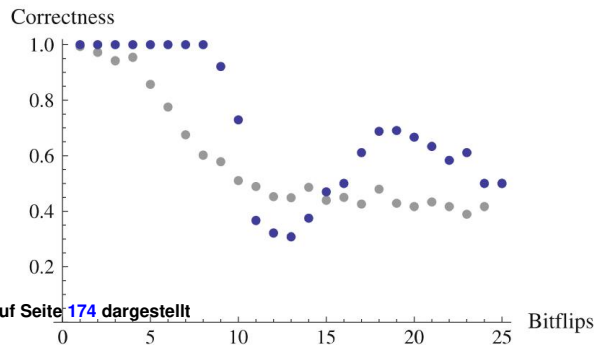
Return(Random(„+“, „-“))

Abb. 8.16 Der K-NEARESTNEIGHBOR -ALGORITHMUS

Abb. 8.17 Relative Korrektheit der

Klassifikation des nächsten Nachbarn als Funktion der Anzahl invertierter Bits. Die Struktur der Kurve mit ihrem Minimum bei 13 und ihrem Maximum bei 19 hängt mit der speziellen Struktur der Trainingsdaten zusammen. Zum Vergleich sind die Werte des Perzeptrons aus Beispiel 8.3 auf Seite 174 dargestellt

grau



Um Fehleinstufungen aufgrund einzelner Ausreißer zu vermeiden, empfiehlt es sich, die Teilungsfläche etwas zu glätten. Dies kann beispielsweise mit dem K-NEARESTNEIGHBOR- Algorithmus in Abb. 8.16 erreicht werden, der eine Mehrheitsentscheidung unter den k nächsten Nachbarn trifft.

Beispiel 8.4 Wir wenden NEARESTNEIGHBOR nun auf Beispiel 8.3 auf Seite 174 an.

Da es sich um binäre Daten handelt, verwenden wir die Hamming-Distanz als Distanzmetrik.⁴ Als Testbeispiel verwenden wir wiederum modifizierte Trainingsbeispiele mit jeweils n aufeinanderfolgenden invertierten Bits. In Abb. 8.17 ist der Prozentsatz korrekt klassifizierter Testbeispiele als Funktion der Anzahl invertierter Bits b dargestellt. Bei bis zu acht invertierten Bits werden alle Muster korrekt erkannt. Ab diesem Zeitpunkt steigt die Zahl der Fehler schnell an. Dies ist nicht überraschend, da das Trainingsmuster Nummer 2 aus Abb. 8.11 auf Seite 174 aus der Klasse M_+ eine Hamming-Distanz von 9 zu den beiden Trainingsbeispielen Nummer 4 und 5 aus der anderen Klasse hat. Dies bedeutet, dass das Testmuster mit hoher Wahrscheinlichkeit den Mustern der anderen Klasse nahe kommt. Wir sehen ganz deutlich, dass die Klassifizierung des nächsten Nachbarn in dieser Anwendung dem Perzeptron für bis zu acht falsche

⁴Der Hamming-Abstand zwischen zwei Bitvektoren ist die Anzahl der unterschiedlichen Bits der beiden Vektoren.

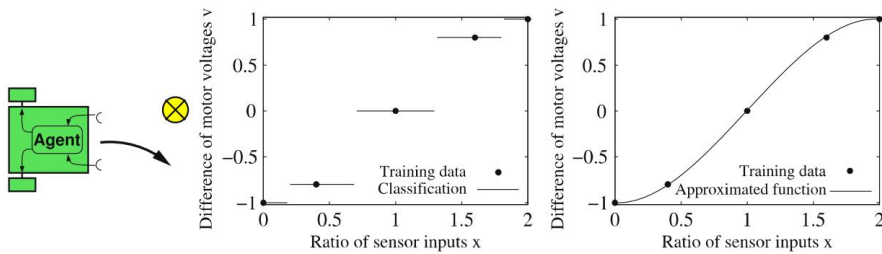


Abb. 8.18 Der Lernagent, der Licht meiden soll (*links*), dargestellt als Klassifikator (*Mitte*) und als Näherungswert (*rechts*)

8.3.1 Zwei Klassen, viele Klassen, Näherung

Die Klassifizierung „Nächster Nachbar“ kann auch auf mehr als zwei Klassen angewendet werden. Genau wie bei zwei Klassen wird die Klasse des zu klassifizierenden Merkmalsvektors einfach als die Klasse des nächsten Nachbarn festgelegt. Für die k -nächste-Nachbarn-Methode ist die Klasse als die Klasse mit den meisten Mitgliedern unter den k nächsten Nachbarn zu bestimmen.

Ist die Anzahl der Klassen groß, ist der Einsatz von Klassifikationsalgorithmen in der Regel nicht mehr sinnvoll, da mit der Anzahl der Klassen auch die Größe der notwendigen Trainingsdaten schnell wächst. Darüber hinaus gehen bei der Klassifizierung vieler Klassen unter bestimmten Umständen wichtige Informationen verloren. Dies wird im folgenden Beispiel deutlich.

Beispiel 8.5 Ein autonomer Roboter mit einfachen Sensoren ähnlich den in Abb. 1.1 auf Seite 2 vorgestellten Braitenberg-Fahrzeugen soll lernen, sich vom Licht zu entfernen. Das heißt, es soll möglichst optimal lernen, seine Sensorwerte auf ein Lenksignal abzubilden, das die Fahrtrichtung steuert. Der Roboter ist auf seiner Vorderseite mit zwei einfachen Lichtsensoren ausgestattet. Aus den beiden Sensorsignalen (mit sl für den linken und sr für den rechten Sensor) wird die Beziehung $x = sr/sl$ berechnet.

Um ab diesem Wert x die Elektromotoren der beiden Räder anzusteuern, ist die Differenz $v = U_r - U_l$ der beiden Spannungen U_r und U_l des linken bzw. rechten Motors.

Die Aufgabe des Lernagenten besteht nun darin, einem Lichtsignal auszuweichen. Es muss also eine 5- Abbildung f lernen, die den „richtigen“ Wert $v = f(x)$ berechnet.

Dazu führen wir ein Experiment durch, bei dem wir für einige Messwerte x einen möglichst optimalen Wert v finden. Diese Werte sind als Datenpunkte in Abb. 8.18 dargestellt und sollen als Trainingsdaten für den Lernagenten dienen. Bei der Klassifizierung des nächsten Nachbarn wird jeder Punkt im Merkmalsraum (d. h. auf der x -Achse) genau wie sein nächster Nachbar in den Trainingsdaten klassifiziert. Die Funktion zur Steuerung der Motoren ist dann eine Schrittfunktion mit großen Sprüngen (Abb. 8.18 Mitte). Wenn wir feinere Schritte wollen, müssen wir entsprechend mehr Trainingsdaten bereitstellen. An

5Um das Beispiel einfach und lesbar zu halten, wurde der Merkmalsvektor x bewusst eins beibehalten dimensional.

Andererseits können wir eine stetige Funktion erhalten, wenn wir eine glatte Funktion annähern, um die fünf Punkte anzupassen (Abb. 8.18 auf Seite 179 rechts). Die Anforderung, dass die Funktion f stetig sein muss, führt zu sehr guten Ergebnissen, auch ohne zusätzliche Datenpunkte.

Für die Approximation von Funktionen an Datenpunkten gibt es viele mathematische Methoden, beispielsweise die Polynominterpolation, die Spline-Interpolation oder die Methode der kleinsten Quadrate. In höheren Dimensionen wird die Anwendung dieser Methoden problematisch. Die besondere Schwierigkeit bei der KI besteht darin, dass modellfreie Approximationsverfahren benötigt werden. Das heißt, eine gute Näherung der Daten muss ohne Kenntnisse über besondere Eigenschaften der Daten oder der Anwendung erfolgen. Sehr gute Ergebnisse wurden hier mit neuronalen Netzen und anderen nichtlinearen Funktionsapproximatoren erzielt, die in Kap. 9.

Die k -nächste-Nachbarn-Methode kann auf einfache Weise auf das Approximationsproblem angewendet werden. Im Algorithmus K-NEARESTNEIGHBOR wird nach der Bestimmung der Menge $V = \{x_1, x_2, \dots, x_k\}$ der durchschnittliche Funktionswert der k nächsten Nachbarn ermittelt

$$\hat{f}(x) = \frac{1}{k} \sum_{i=1}^k f(x_i) \quad (8.2)$$

wird berechnet und als Näherung $\hat{f}$ für den Abfragevektor x verwendet. Je größer k wird, desto glatter ist die Funktion $\hat{f}$.

8.3.2 Entfernung ist relevant

Bei der praktischen Anwendung sowohl diskreter als auch kontinuierlicher Varianten der k -Nächste-Nachbarn-Methode treten häufig Probleme auf. Wenn k größer wird, gibt es typischerweise mehr Nachbarn mit großem Abstand als solche mit kleinem Abstand. Dabei wird die Berechnung von $\hat{f}$ von weit entfernten Nachbarn dominiert. Um dies zu verhindern, werden die k Nachbarn so gewichtet, dass die weiter entfernten Nachbarn einen geringeren Einfluss auf das Ergebnis haben. Bei der Mehrheitsentscheidung im Algorithmus K-NEARESTNEIGHBOR werden die „Stimmen“ mit dem Gewicht gewichtet

$$w_i = \frac{1}{1 + \tilde{y} d(x, x_i)^2}, \quad (8.3)$$

die mit dem Quadrat der Entfernung abnimmt. Die Konstante $\tilde{y}$ bestimmt die Geschwindigkeit der Abnahme der Gewichte. Gleichung (8.2) wird nun ersetzt durch

$$\hat{f}(x) = \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}.$$

Bei einer gleichmäßig verteilten Punktkonzentration im Merkmalsraum wird dadurch sichergestellt, dass der Einfluss der Punkte mit zunehmendem Abstand asymptotisch gegen Null geht. Dadurch wird es möglich, viele oder sogar alle Trainingsdaten zu verwenden, um einen gegebenen Eingabevektor zu klassifizieren oder anzunähern.

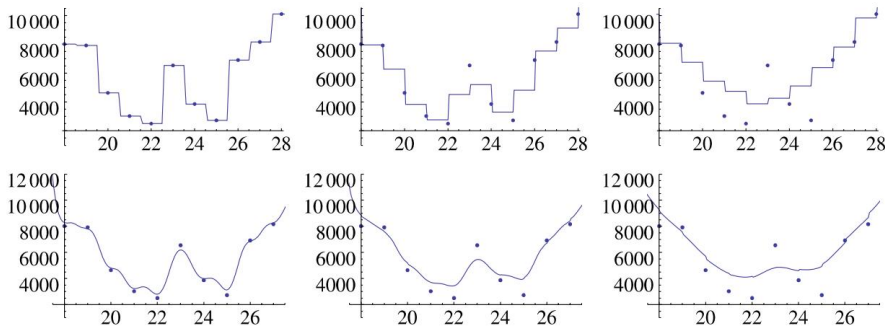


Abb. 8.19 Vergleich der k -Nearest-Neighbor-Methode (obere Reihe) mit $k = 1$ (links), $k = 2$ (Mitte) und $k = 6$ (rechts) zu ihrer abstandsgewichteten Variante (untere Reihe) mit $\gamma = 20$ (links), $\gamma = 4$ (Mitte) und $\gamma = 1$ (rechts) für einen eindimensionalen Datensatz

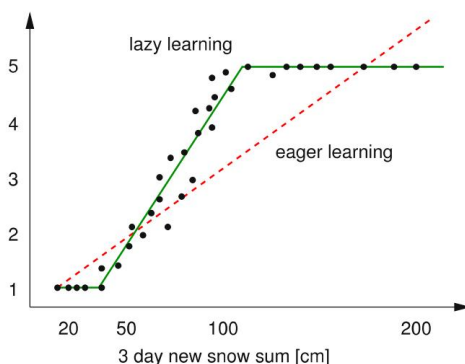
Um ein Gefühl für diese Methoden zu bekommen, wird in Abb. 8.19 die k -Nearest-Neighbor-Methode (in der oberen Reihe) mit ihrer abstandsgewichteten Optimierung verglichen. Aufgrund der Mittelung können beide Methoden verallgemeinern, also Rauschen auslöschen, wenn die Anzahl der Nachbarn für k -nächster Nachbar oder der Parameter γ entsprechend eingestellt ist. Die Diagramme zeigen gut, dass die abstandsgewichtete Methode eine viel glattere Näherung liefert als die k -nächste Nachbarmethode. Hinsichtlich der Approximationsqualität kann dieses sehr einfache Verfahren gut mit anspruchsvollen Approximationsalgorithmen wie nichtlinearen neuronalen Netzen, Support-Vektor-Maschinen und Gauß-Prozessen mithalten.

Es gibt viele Alternativen zu der in (8.3) auf Seite 180 angegebenen Gewichtsfunktion (auch Kernel genannt). Beispielsweise kann eine Gauß-Funktion oder eine ähnliche glockenförmige Funktion verwendet werden. Bei den meisten Anwendungen sind die Ergebnisse bei der Auswahl des Kernels nicht sehr aussagekräftig. Allerdings hat der Breitenparameter γ , der bei all diesen Funktionen manuell eingestellt werden muss, großen Einfluss auf die Ergebnisse, wie in Abb. 8.19 dargestellt. Um eine solche umständliche manuelle Anpassung zu vermeiden, wurden Optimierungsverfahren zur automatischen Einstellung dieses Parameters entwickelt [SA94, SE10].

8.3.3 Rechenzeiten

Wie bereits erwähnt, erfolgt das Training in allen Varianten der Nearest-Neighbor-Methode durch einfaches Speichern aller Trainingsvektoren zusammen mit ihren Labels (Klassenwerten) oder dem Funktionswert $f(x)$. Daher gibt es keinen anderen Lernalgorithmus, der so schnell lernt. Allerdings kann die Beantwortung einer Anfrage zur Klassifizierung oder Approximation eines Vektors x sehr teuer werden. Allein das Finden der k nächsten Nachbarn für n Trainingsdaten erfordert einen Aufwand, der linear mit n wächst. Für die Klassifizierung bzw. Approximation kommt zusätzlich ein Aufwand hinzu, der linear in k ist. Die Gesamtrechenzeit wächst somit mit $\gamma(n + k)$. Bei großen Mengen an Trainingsdaten kann dies zu Problemen führen.

Abb. 8.20 Zur Bestimmung der Lawinengefahr wird eine Funktion aus Trainingsdaten approximiert. Hier zum Vergleich ein lokales Modell (*durchgezogene Linie*) und ein globales Modell (*gestrichelte Linie*).
avalanche level



8.3.4 Zusammenfassung und Ausblick

Da in der Lernphase der vorgestellten Nearest-Neighbor-Methoden nichts passiert, werden solche Algorithmen auch als *Lazy Learning bezeichnet*, im Gegensatz zum *Eager Learning*, bei dem die Lernphase teuer sein kann, die Anwendung auf neue Beispiele aber sehr effizient ist. Das Perzeptron und alle anderen neuronalen Netze, das Lernen von Entscheidungsbäumen und das Lernen von Bayes'schen Netzen sind Methoden des eifrigen Lernens. Da die Lazy-Learning-Methoden zur Annäherung an eine neue Eingabe Zugriff auf den Speicher mit allen Trainingsdaten benötigen, werden sie auch als *gedächtnisbasiertes L*

Um diese beiden Klassen von Lernprozessen zu vergleichen, verwenden wir als Beispiel die Aufgabe, aus der Neuschneemenge in einem bestimmten Gebiet der Schweiz die aktuelle Lawinengefahr zu ermitteln.⁶ In Abb. 8.20 sind von Experten ermittelte Werte eingetragen, die wir wollen als Trainingsdaten verwenden. Bei der Anwendung eines Eager-Learning-Algorithmus, der eine lineare Approximation der Daten vornimmt, wird die in der Abbildung dargestellte gestrichelte Linie berechnet. Aufgrund der Beschränkung auf eine Gerade ist der Fehler mit maximal etwa 1,5 Gefährdungsstufen relativ groß. Beim Lazy Learning wird nichts berechnet, bevor eine Abfrage nach der aktuellen Gefährdungsstufe eintrifft. Dann wird die Antwort aus mehreren nächsten Nachbarn, also lokal, berechnet. Dadurch könnte die in der Abbildung gezeigte Kurve entstehen, die aus Liniensegmenten zusammengesetzt ist und deutlich kleinere Fehler aufweist. Der Vorteil der Lazy-Methode ist ihre Lokalität. Die Näherung erfolgt lokal aus der aktuellen Neuschneehöhe und nicht global. Für grundsätzlich gleiche Klassen von Funktionen (z. B. lineare Funktionen) sind daher die Lazy-Algorithmen besser.

Nearest-Neighbor-Methoden eignen sich gut für alle Problemsituationen, in denen eine gute lokale Näherung erforderlich ist, die aber keine hohen Anforderungen an die Geschwindigkeit des Systems stellen. Der hier erwähnte Lawinenvorhersager, der einmal täglich läuft, ist eine solche Anwendung. Nearest-Neighbor-Methoden sind nicht geeignet, wenn eine Beschreibung des aus den Daten extrahierten Wissens für den Menschen verständlich sein muss, was heute bei vielen Data-Mining-Anwendungen der Fall ist (siehe Abschn. 8.4).

⁶Die dreitägige Gesamtschneemenge ist tatsächlich ein wichtiges Merkmal zur Bestimmung der Gefährdungsstufe. In der Praxis kommen jedoch noch weitere Attribute zum Einsatz [Bra01]. Das hier verwendete Beispiel ist vereinfacht.

Besonderheit	Abfrage	Koffer aus Kofferbasis
Defektes Teil:	Rücklicht	Vorderlicht
Fahrradmodell:	Marin Pine Mountain	VSF T400
Jahr:	1993	2001
Energiequelle:	Batterie	Dynamo
Zustand der Glühbirne:	ok	OK
Zustand des Lichtkabels:	?	OK
Lösung		
Diagnose:	?	Elektrischer Kontakt vorn fehlt
Reparatur:	?	Elektrischen Kontakt vorn herstellen

Abb. 8.21 Einfaches Diagnosebeispiel für eine Abfrage und den zugehörigen Fall aus der Fallbasis

In den letzten Jahren erfreuen sich diese gedächtnisbasierten Lernmethoden immer größerer Beliebtheit, und es wurden verschiedene verbesserte Varianten (z. B. lokal gewichtete lineare Regression) entwickelt angewendet [Cle79].

Um die beschriebenen Methoden nutzen zu können, müssen die Trainingsdaten verfügbar sein in Form von Vektoren aus ganzen Zahlen oder reellen Zahlen. Sie sind daher ungeeignet für Anwendungen, in denen die Daten symbolisch dargestellt werden, beispielsweise als erstes Ordnungslogikformeln. Darauf wollen wir nun kurz eingehen.

8.3.5 Fallbasiertes Denken

Beim Case-based Reasoning (CBR) wird die Nearest-Neighbor-Methode auf symbolische Problembeschreibungen und deren Lösungen erweitert. CBR wird zur Diagnose technischer Probleme im Kundendienst oder bei Telefon-Hotlines eingesetzt. Das gezeigte Beispiel

In Abb. 8.21 über die Diagnose eines Fahrradlichtausfalls wird dieser Typ veranschaulicht Situation.

Für die Anfrage eines Kunden mit einem defekten Hinterrad wird nach einer Lösung gesucht Licht. In der rechten Spalte ist ein Fall angegeben, der der Abfrage in der mittleren Spalte ähnelt. Dies ergibt sich aus der Fallbasis, die den Trainingsdaten am nächsten kommt Nachbarmethode. Wenn wir einfach den ähnlichsten Fall nehmen würden, wie wir es im nächsten tun Nachbarmethode, dann würden wir am Ende versuchen, das Frontlicht zu reparieren, wenn das Rücklicht ist kaputt. Wir benötigen daher eine Rücktransformation der Lösung des entdeckten ähnlichen Problems zurück in die Abfrage. Die wichtigsten Schritte zur Lösung auf einen CBR-Fall werden in Abb. 8.22 auf Seite 184 durchgeführt. Die Transformation in diesem Das Beispiel ist einfach: Das Rücklicht wird dem Vorderlicht zugeordnet.

So schön und einfach diese Methode in der Theorie auch erscheint, in der Praxis ist der Aufbau von CBR-Diagnosesystemen eine sehr schwierige Aufgabe. Die drei Hauptschwierigkeiten Sind:

Modellierung Die Domänen der Anwendung müssen in einem formalen Kontext modelliert werden.

Hier kommt es zur logischen Monotonie, die wir aus Kap. 4, bereitet Schwierigkeiten. Kann der Entwickler alle möglichen Sonderfälle und Problemvarianten vorhersagen und abbilden?

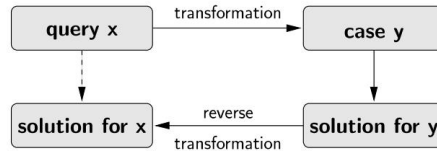


Abb. 8.22 Wenn für einen Fall x ein ähnlicher Fall y gefunden wird, muss, um eine Lösung für x zu erhalten, die Transformation bestimmt und ihre Umkehrung auf den entdeckten Fall y angewendet werden

Ähnlichkeit Finden einer geeigneten Ähnlichkeitsmetrik für symbolische, nicht numerische Features.

Transformation Auch wenn ein ähnlicher Fall gefunden wird, ist noch nicht klar, wie die Transformationsabbildung und ihre Umkehrung aussehen sollen.

Tatsächlich sind heute praktische CBR-Systeme für diagnostische Anwendungen im Einsatz. Allerdings bleiben diese aus den genannten Gründen hinsichtlich Leistung und Flexibilität weit hinter menschlichen Experten zurück. Eine interessante Alternative zu CBR sind die in Kap. vorgestellten Bayes'schen Netze. 7. Oft lässt sich die symbolische Problemdarstellung auch recht gut auf diskrete oder kontinuierliche numerische Merkmale abbilden. Dann können die genannten induktiven Lernmethoden wie Entscheidungsbäume oder neuronale Netze erfolgreich eingesetzt werden.

8.4 Entscheidungsbaum-Lernen

Das Lernen von Entscheidungsbäumen ist ein außerordentlich wichtiger Algorithmus für die KI, da er sehr leistungsfähig, aber auch einfach und effizient ist, um Wissen aus Daten zu extrahieren. Gegenüber den beiden bereits vorgestellten Lernalgorithmen weist es einen wichtigen Vorteil auf. Das extrahierte Wissen ist nicht nur als Black-Box-Funktion verfügbar und nutzbar, sondern kann in Form eines lesbaren Entscheidungsbaums auch vom Menschen leicht verstanden, interpretiert und kontrolliert werden. Dies macht das Lernen von Entscheidungsbäumen auch zu einem wichtigen Werkzeug für das Data Mining.

Wir werden Funktion und Anwendung des Entscheidungsbaumlernens mithilfe des C4.5-Algorithmus diskutieren. C4.5 wurde 1993 vom Australier Ross Quinlan eingeführt und ist eine Verbesserung seines Vorgängers ID3 (Iterative Dichotomiser 3, 1986). Es ist für die nichtkommerzielle Nutzung frei verfügbar [Qui93]. Eine Weiterentwicklung, die noch effizienter arbeitet und die Kosten von Entscheidungen berücksichtigen kann, ist C5.0 [Qui93].

Das von Leo Breiman [BFOS84] entwickelte CART- System (Classification and Regression Trees, 1984) funktioniert ähnlich wie C4.5. Es verfügt über eine praktische grafische Benutzeroberfläche, ist aber sehr teuer.

Zwanzig Jahre zuvor, im Jahr 1964, wurde von J. Sonquist und J. Morgan das CHAID-System (Chi-square Automatic Interaction Detectors) eingeführt, das automatisch Entscheidungsbäume generieren kann. Es hat die bemerkenswerte Eigenschaft, dass es das Wachstum des Baumes stoppt, bevor er zu groß wird, aber heute hat es keine Bedeutung mehr.

Tabelle 8.4 Variablen für das Ski-Klassifizierungsproblem

Variable	Wert	Beschreibung
<i>Ski</i> (Zielvariabel)	Ja Nein	Soll ich zum nächsten Skigebiet fahren? mit genügend Schnee?
<i>Sonne</i> (Feature)	Ja Nein	Gibt es heute Sonnenschein?
<i>Snow_Dist</i> (Funktion)	≤100, >100	Entfernung zum nächstgelegenen Skigebiet mit Gute Schneeverhältnisse (über/unter). 100 km)
<i>Wochenende</i> (Feature)	Ja Nein	Ist heute Wochenende?

Interessant ist auch das Data-Mining-Tool *KNIME* (Konstanz Information Miner), das über eine benutzerfreundliche Oberfläche verfügt und mithilfe der *WEKA*-Java-Bibliothek auch macht Induktion von Entscheidungsbäumen möglich. In Abschn. 8.8 werden wir *KNIME* vorstellen.

Nun zeigen wir zunächst an einem einfachen Beispiel, wie ein Entscheidungsbaum konstruiert werden kann aus Trainingsdaten, um dann den Algorithmus zu analysieren und auf weitere anzuwenden komplexes LEXMED-Beispiel für die medizinische Diagnose.

8.4.1 Ein einfaches Beispiel

Ein begeisterter Skifahrer, der in der Nähe der High Sierra, einer wunderschönen Bergkette in Kalifornien, lebt, möchte einen Entscheidungsbaum, der ihm bei der Entscheidung hilft, ob es sich lohnt, mit dem Auto zu seinem Auto zu einem Skigebiet in den Bergen. Wir haben also einen Zwei-Klassen-Problemski *ja/nein* basierend auf den in Tabelle 8.4 aufgeführten Variablen.

Abbildung 8.23 auf Seite 186 zeigt einen *Entscheidungsbaum* für dieses Problem. Ein Entscheidungsbaum ist ein Baum, dessen innere Knoten Merkmale (Attribute) darstellen. Jede Kante steht für ein Attributwert. An jedem Blattknoten wird ein Klassenwert angegeben.

Die für die Konstruktion des Entscheidungsbaums verwendeten Daten sind in Tabelle 8.5 aufgeführt Seite 186. Jede Zeile in der Tabelle enthält die Daten für einen Tag und stellt somit einen dar Probe. Bei näherer Betrachtung erkennen wir, dass Zeile 6 und Zeile 7 einander widersprechen. Daher kann kein deterministischer Klassifizierungsalgorithmus alle Daten korrekt klassifizieren. Die Anzahl der falsch klassifizierten Daten muss daher ≥1 sein. Der Baum in Abb. 8.23 auf Seite 186 klassifiziert die Daten somit optimal.

Wie entsteht aus den Daten ein solcher Baum? Um diese Frage zu beantworten, werden wir unter Beschränken wir uns zunächst auf diskrete Attribute mit endlich vielen Werten. Weil das Die Anzahl der Attribute ist ebenfalls endlich und jedes Attribut kann höchstens einmal pro Pfad vorkommen. Es gibt endlich viele verschiedene Entscheidungsbäume. Ein einfacher, offensichtlicher Algorithmus für die Die Konstruktion eines Baumes würde einfach alle Bäume generieren und dann für jeden Baum berechnen die Anzahl der fehlerhaften Klassifizierungen der Daten und wählen Sie am Ende den Baum aus mit der minimalen Fehleranzahl. Somit hätten wir sogar einen optimalen Algorithmus (im Sinne von Fehlern für die Trainingsdaten) für das Lernen im Entscheidungsbaum.

Der offensichtliche Nachteil dieses Algorithmus ist seine unzumutbar hohe Rechenzeit, sobald die Anzahl der Attribute etwas größer wird. Wir werden Entwickeln Sie nun einen heuristischen Algorithmus, der ausgehend von der Wurzel rekursiv aufbaut

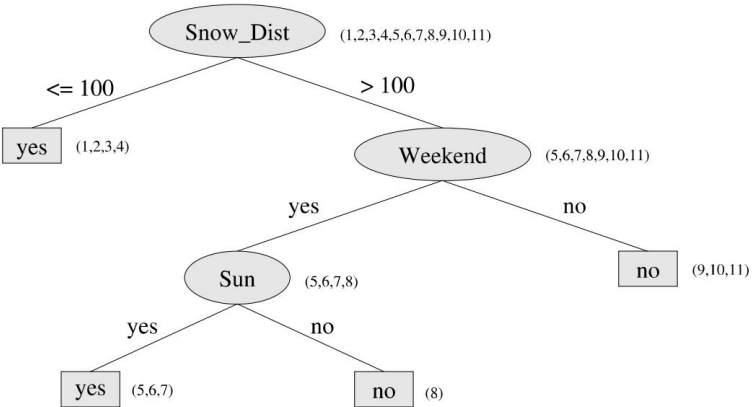


Abb. 8.23 Entscheidungsbaum für das Ski-Klassifizierungsproblem. In den Listen rechts *neben* den Knoten: die Nummern der entsprechenden Trainingsdaten sind angegeben. Beachten Sie, dass bei den Blattknoten *sonnig = ja* ist nur zwei der drei Beispiele sind korrekt klassifiziert

Tabelle 8.5 Datensatz für die Klassifizierungsproblem im Skisport

Tag	Snow_Dist	Wochenende	Sonnenskilauf
1	≤100	Ja	Ja ja
2	≤100	Ja	Ja ja
3	≤100	Ja	Nein Ja
4	≤100	NEIN	Ja ja
5	>100	Ja	Ja ja
6	>100	Ja	Ja ja
7	>100	Ja	ja Nein
8	>100	Ja	Nein Nein
9	>100	NEIN	ja Nein
10	>100	NEIN	ja Nein
11	>100	NEIN	Nein Nein

ein Entscheidungsbaum. Zuerst wird das Attribut mit dem höchsten Informationsgewinn (*Snow_Dist*) ausgewählt aus der Menge aller Attribute für den Wurzelknoten ausgewählt. Für jeden Attributwert (≤100, >100) gibt es einen Ast im Baum. Nun wird dieser Vorgang für jeden Zweig wiederholt rekursiv. Bei der Generierung der Knoten wird immer das Attribut mit dem höchsten Informationsgewinn unter den noch nicht genutzten Attributen ausgewählt
Geist einer gierigen Strategie.

8.4.2 Entropie als Maß für Informationsinhalte

Der beschriebene Top-Down-Algorithmus für die Konstruktion eines Entscheidungsbaums Schritt wählt das Attribut mit dem höchsten Informationsgewinn aus. Wir stellen jetzt vor

Entropie als *Maß* für den Informationsgehalt eines Trainingsdatensatzes D . Wenn wir im obigen **Beispiel** nur die binäre Variable **Skating** betrachten, dann kann D beschrieben werden als

$$D = (\text{ja}, \text{ja}, \text{ja}, \text{ja}, \text{ja}, \text{ja}, \text{nein}, \text{nein}, \text{nein}, \text{nein})$$

mit geschätzten Wahrscheinlichkeiten

$$p_1 = P(\text{ja}) = 6/11 \text{ und } p_2 = P(\text{nein}) = 5/11.$$

Hier haben wir offensichtlich eine Wahrscheinlichkeitsverteilung $p = (6/11, 5/11)$. Im Allgemeinen lautet dies für ein n -Klassen-Problem

$$p = (p_1, \dots, p_n)$$

mit

$$\sum_{i=1}^n p_i = 1.$$

Um den Informationsgehalt einer Verteilung einzuführen, beobachten wir zwei Extremfälle.

Erst mal lassen

$$p = (1, 0, 0, \dots, 0). \quad (8.4)$$

Das heißt, das erste der n Ereignisse wird mit Sicherheit eintreten und alle anderen nicht. Die Unsicherheit über den Ausgang der Ereignisse ist daher minimal. Im Gegensatz dazu für die Gleichverteilung

$$p = \left(\frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \right) \quad (8.5)$$

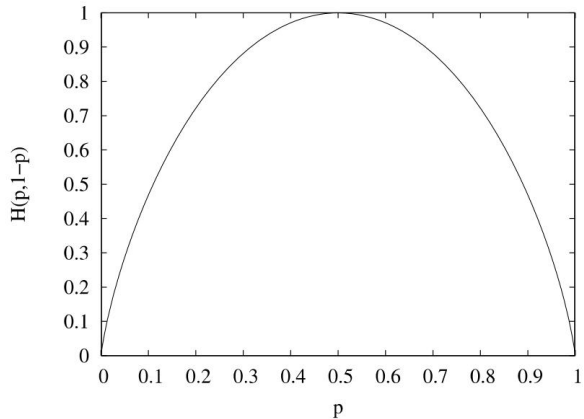
Die Unsicherheit ist maximal, da kein Ereignis von den anderen unterschieden werden kann. Hier stellte sich Claude Shannon die Frage, wie viele Bits nötig wären, um ein solches Ereignis zu kodieren. Im bestimmten Fall von (8.4) werden Nullbits benötigt, da wir wissen, dass der Fall $i = 1$ immer auftritt. Im gleichverteilten Fall von (8.5) gibt es n gleichwahrscheinliche Möglichkeiten. Für binäre Kodierungen werden hier $\log_2 n$ Bits benötigt. Da alle Einzelwahrscheinlichkeiten $p_i = 1/n$ sind, werden für diese Kodierung $\log_2$ -Bits benötigt.

Im allgemeinen Fall $p = (p_1, \dots, p_n)$ wird, wenn die Wahrscheinlichkeiten der Elementarereignisse von der Gleichverteilung abweichen, der Erwartungswert H für die Anzahl der Bits berechnet. Dazu gewichten wir alle Werte $\log_2 p_i$ mit ihren Wahrscheinlichkeiten und erhalten

$$H = - \sum_{i=1}^n p_i (\log_2 p_i) = - \sum_{i=1}^n p_i \log_2 p_i.$$

Je mehr Bits wir zum Kodieren eines Ereignisses benötigen, desto größer ist eindeutig die Unsicherheit über das Ergebnis. Deshalb definieren wir:

Abb. 8.24 Die Entropiefunktion für den Fall zweier Klassen. Wir sehen das Maximum bei $p = 1/2$ und die Symmetrie hinsichtlich der Vertauschung von p und $1 - p$



Definition 8.4 Die *Entropie* H als Maß für die Unsicherheit einer Wahrscheinlichkeitsverteilung ist definiert durch⁷

$$H(p) = H(p_1, \dots, p_n) := - \sum_{i=1}^n p_i \log_2 p_i.$$

Eine detaillierte Herleitung dieser Formel findet sich in [SW76]. Wenn wir das bestimmte Ereignis $p = (1, 0, \dots, 0)$, dann $0 \log_2 0$ einsetzen, ergibt sich ein undefinierter Ausdruck. Wir lösen dieses Problem durch die Definition $0 \log_2 0 := 0$ (siehe Aufgabe 8.9 auf Seite 218).

Jetzt können wir $H(1, 0, \dots, 0) = 0$ berechnen. Wir werden zeigen, dass die Entropie in der Hypercube $[0, 1]^N$ unter der Einschränkung $\sum_{i=1}^N p_i = 1$ nimmt seinen Maximalwert an). Im mit gleichmäßiger Verteilung $\frac{1}{N}, \dots, \frac{1}{N}$ Falle einer Veranstaltung mit zwei möglichen (Ergebnisse, die zwei Klassen entsprechen, ist das Ergebnis

$$H(p) = H(p_1, p_2) = H(p_1, 1 - p_1) = -(p_1 \log_2 p_1 + (1 - p_1) \log_2 (1 - p_1)).$$

Dieser Ausdruck ist als Funktion von p_1 in Abb. 8.24 dargestellt, wobei sein Maximum bei $p_1 = 1/2$ liegt.

Da jedem klassifizierten Datensatz D durch Schätzung der Klassenwahrscheinlichkeiten eine Wahrscheinlichkeitsverteilung p zugewiesen wird, können wir das Konzept der Entropie auf Daten erweitern

⁷ In (7.9) auf Seite 124 wird zur Definition der Entropie der natürliche Logarithmus anstelle von $\log_2$ verwendet. Da hier, wie auch bei der MaxEnt-Methode, nur Entropien verglichen werden, spielt dieser Unterschied keine Rolle. (siehe Übung 8.11 auf Seite 218).

die Definition

$$H(D) = H(p).$$

Da nun der Informationsgehalt $I(D)$ des Datensatzes D das Gegenteil von Unsicherheit sein soll. So definieren wir:

Definition 8.5 Der Informationsgehalt eines Datensatzes ist definiert als

$$I(D) := 1 - H(D). \quad (8.6)$$

8.4.3 Informationsgewinn

Wenn wir die Entropieformel auf das Beispiel anwenden, ist das Ergebnis

$$H(6/11, 5/11) = 0,994$$

Beim Aufbau eines Entscheidungsbaums wird der Datensatz nach jedem neuen Attribut weiter unterteilt. Je mehr ein Attribut den Informationsgehalt der Verteilung durch Aufteilung der Daten erhöht, desto besser ist dieses Attribut. Wir definieren entsprechend:

Definition 8.6 Der Informationsgewinn $G(D, A)$ durch die Verwendung des Attributs A wird durch die Differenz des durchschnittlichen Informationsgehalts des Datensatzes $D = D_1 \dot{\cup} D_2 \dot{\cup} \dots \dot{\cup} D_n$ dividiert durch das n -Wert-Attribut A bestimmt und der Informationsgehalt $I(D)$ des ungeteilten Datensatzes, der ergibt

$$G(D, A) = \sum_{i=1}^N \frac{|D_i|}{|D|} I(D_i) - I(D).$$

Mit (8.6) erhalten wir daraus

$$\begin{aligned} G(D, A) &= \sum_{i=1}^N \frac{|D_i|}{|D|} I(D_i) - I(D) = \sum_{i=1}^N \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D)) \\ &= 1 - \sum_{i=1}^N \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^N \frac{|D_i|}{|D|} H(D_i). \end{aligned} \quad (8.7)$$

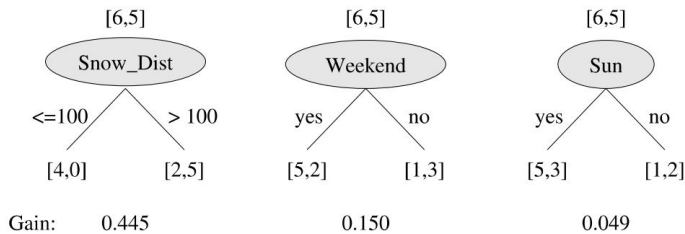


Abb. 8.25 Der berechnete Gewinn für die verschiedenen Attribute spiegelt wider, ob die Division der Daten durch das jeweilige Attribut zu einer besseren Klasseneinteilung führt. Je stärker die durch das Attribut erzeugten Verteilungen von der Gleichverteilung abweichen, desto höher ist der Informationsgewinn

Auf unser Beispiel für das Attribut *Snow_Dist* angewendet ergibt dies

$$\begin{aligned}
 G(D, \text{Snow\_Dist}) &= H(D) - \left(\frac{4}{11} H(D \leq 100) + \frac{7}{11} H(D > 100) \right) \\
 &= 0,994 - \left(\frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0,863 \right) = 0,445
 \end{aligned}$$

Analog erhalten wir

$$G(D, \text{Wochenende}) = 0,150$$

Und

$$G(D, \text{Sonne}) = 0,049.$$

Das Attribut *Snow_Dist* wird nun zum Wurzelknoten des Entscheidungsbaums. Die Situation der Auswahl dieses Attributs wird in Abb. 8.25 noch einmal verdeutlicht.

Die beiden Attributwerte ≤ 100 und > 100 erzeugen zwei Kanten im Baum, die den Teilmengen $D \leq 100$ und $D > 100$ entsprechen. Für die Teilmenge $D \leq 100$ ist die Klassifizierung eindeutig *Ja*, somit endet der Baum hier. Im anderen Zweig $D > 100$ gibt es kein eindeutiges Ergebnis. Somit wiederholt sich der Algorithmus rekursiv. Von den beiden noch verfügbaren Attributen *So* und *Wochenende* muss das bessere ausgewählt werden. Wir berechnen

$$G(D > 100, \text{Wochenende}) = 0,292$$

Und

$$G(D > 100, \text{Sonne}) = 0,170.$$

Der Knoten bekommt somit das Attribut *Wochenende* zugewiesen. Für *Weekend* = *no* endet der Baum mit der Entscheidung *Ski* = *no*. Eine Berechnung der Verstärkung ergibt hier den Wert 0. Für *Weekend* = *ja*, *So* ergibt sich ein Gewinn von 0,171. Dann bricht der Aufbau des Baums ab, da keine weiteren Attribute mehr vorhanden sind, obwohl Beispiel Nummer 7 falsch klassifiziert ist. Den fertigen Baum kennt man bereits aus Abb. 8.23 auf Seite 186.

8.4.4 Anwendung von C4.5

Der gerade generierte Entscheidungsbaum kann auch von C4.5 generiert werden. Die Trainingsdaten werden in einer Datendatei ski.data im folgenden Format gespeichert:

```
<=100, ja, ja, ja <=100, ja, ja <=100,
ja, nein, ja <=100, nein, ja, ja >100, ja,
ja, ja >100, ja, ja, ja >100, ja, ja, nein
>100, ja, nein, nein >100, nein, ja,
nein >100, nein, ja, nein >100, nein,
```

nein, nein

Die Informationen zu Attributen und Klassen werden in der Datei ski.names gespeichert (Zeilen, die mit „|“ beginnen, sind Kommentare):

```
|Unterricht: nein: nicht Skifahren, ja: Skifahren gehen |
```

```
Nein Ja.
```

```
|
```

```
|Attribute |
```

```
Snow_Dist:          <=100,>100.
```

```
Wochenende:        Nein Ja.
```

```
Sonne:             Nein Ja.
```

C4.5 wird dann über die Unix-Befehlszeile aufgerufen und generiert den unten gezeigten Entscheidungsbaum, der mithilfe von Einrückungen formatiert ist. Die Option -f gibt den Namen der Eingabedatei an und die Option -m gibt die Mindestanzahl an Trainingsdatenpunkten an, die zum Generieren eines neuen Zweigs im Baum erforderlich sind. Da die Anzahl der Trainingsdatenpunkte in diesem Beispiel äußerst gering ist, ist -m 1 hier sinnvoll. Für größere Datensätze sollte ein Wert von mindestens -m 10 verwendet werden.


```
unixprompt> c4.5 -f ski -m 1
```

C4.5 [Release 8] Entscheidungsbaumgenerator

Mi, 23. August 10:44:49 2010

Optionen:

Dateistamm <Ski>

Sinnvoller Test erfordert 2 Zweige mit ≥ 1 Fällen

Lesen Sie 11 Fälle (3 Attribute) aus ski.data

Entscheidungsbaum:

Snow_Dist = ≤ 100 : ja (4,0)

Snow_Dist = > 100 :

| Wochenende = nein: nein (3,0)

| Wochenende = ja:

| | Sonne = nein: nein (1,0)

| | Sonne = ja: ja (3,0/1,0)

Vereinfachter Entscheidungsbaum:

Snow_Dist = ≤ 100 : ja (4,0/1,2)

Snow_Dist = > 100 : nein (7,0/3,4)

Auswertung der Trainingsdaten (11 Items):

Vor dem Beschneiden		Nach dem Beschneiden		
Größe	Fehler	Größe	Fehler	Schätzen
7	1(9,1%)	3	2(18,2%)	(41,7 %) <<

Zusätzlich wird ein vereinfachter Baum mit nur einem Attribut angegeben. Dieser Baum, der durch Beschneiden entstanden ist (siehe Abschn. 8.4.7), wird in zunehmendem Maße wichtig sein von Trainingsdaten. In diesem kleinen Beispiel macht es noch nicht viel Sinn. Der Fehler Die Rate für beide Bäume in den Trainingsdaten ist ebenfalls angegeben. Die Zahlen in Klammern Geben Sie nach den Entscheidungen die Größe des zugrunde liegenden Datensatzes und die Anzahl der Fehler an. Beispielsweise zeigt die Zeile Sun = ja: ja (3,0/1,0) im oberen Baum an Für diesen Blattknoten Sun = ja gibt es drei Trainingsbeispiele, eines davon ist falsch eingestuft. Daraus kann der Nutzer also ablesen, ob die Entscheidung statistisch fundiert und/oder sicher ist.

In Abb. 8.26 auf Seite 193 können wir nun das Schema des Lernalgorithmus wiedergeben zur Generierung eines Entscheidungsbaums.

Wir kennen nun die Grundlagen der automatischen Entscheidungsgenerierung Bäume. Für die praktische Anwendung sind jedoch wichtige Erweiterungen erforderlich. Wir wird diese anhand der bereits bekannten LECMED- Anwendung einführen .

GENERATEDECISIONTREE(Daten,Knoten)

Amax = Attribut mit maximalem Informationsgewinn Wenn

G(Amax) = 0,

dann wird *der Knoten* zum Blattknoten mit der häufigsten Klasse in *den*

Daten . Andernfalls weisen Sie dem *Knoten*

das Attribut **Amax** zu . Generieren Sie für jeden Wert

a1,...,an von **Amax** einen Nachfolgeknoten: **K1,...,Kn**

Teilen Sie *Daten* in **D1,...,Dn** mit **Di** = {**x** | **Daten**|**Amax**(**x**) = **ai**}

Für alle **i** ∈ {1,...,n}

Wenn alle **x** ∈ **Di** zur gleichen Klasse **Ci** gehören

Generieren Sie dann den Blattknoten **Ki** der Klasse **Ci**

Else **GENERATEDECISIONTREE(Di, Ki)**

Abb. 8.26 Algorithmus zur Konstruktion eines Entscheidungsbaums

8.4.5 Lernen der Blinddarmentzündungsdiagnose

Im Forschungsprojekt LEXMED wurde auf Basis einer Datenbank mit Patientendaten ein Expertensystem zur Diagnose der Blinddarmentzündung entwickelt [ES99, SE00]. Das System, das mit der Methode der maximalen Entropie arbeitet, wird in Abschn. 2.1 beschrieben. 7.3

Wir verwenden nun die LEXMED- Datenbank, um einen Entscheidungsbaum für die Diagnose einer Blinddarmentzündung mit C4.5 zu erstellen. Die als Attribute verwendeten Symptome sind in der Datei `app.names` definiert:

|Definition der Klassen und Attribute | |Klassen

0=Blinddarmentzündung negativ | 1=Appendizitis

positiv 0,1. | |Attribute | Alter:

ununterbrochen.

Sex_(1=m__2=w): 1,2.

Pain_Quadrant1_(0=nein__1=ja): 0,1.

Pain_Quadrant2_(0=nein__1=ja): 0,1.

Pain_Quadrant3_(0=nein__1=ja): 0,1.

Pain_Quadrant4_(0=nein__1=ja): 0,1.

Local_guarding_(0=nein__1=ja): 0,1.

Generalized_guarding_(0=nein__1=ja): 0,1.

Rebound_tenderness_(0=nein__1=ja): 0,1.

Pain_on_tapping_(0=nein__1=ja): 0,1.

Pain_during_rectal_examination_(0=nein__1=ja): 0,1.
 Temp_axial: kontinuierlich.
 Temp_rectal: kontinuierlich.
 Leukozyten: kontinuierlich.
 Diabetes_mellitus_(0=nein__1=ja): 0,1

Wir sehen, dass neben vielen binären Merkmalen wie den verschiedenen Schmerzsymptomen auch kontinuierliche Symptome wie Alter und Fiebertemperatur auftreten. In der folgenden Trainingsdatendatei, `app.data`, wird in jeder Zeile ein Fall beschrieben. In der ersten Zeile steht ein 19-jähriger männlicher Patient mit Schmerzen im dritten Quadranten (unten rechts, wo sich der Blinddarm befindet), den beiden Fieberwerten 36,2 und 37,8 Grad Celsius, einem Leukozytenwert von 13400 und einer positiven Diagnose ist ein entzünd

```
19,1,0,0,1,0,1,0,1,1,0,362,378,13400,0,1
13,1,0,0,1,0,1,0,1,1,1,383,385,18100,0,1
32,2,0,0,1,0,1,0,1,1,0,364,374,11800,0,1 18,2,0,0,1,1,0,0,0,0
,0,362,370,09300,0,0 73,2,1,0,1,1,1,0,1,1,1,376,380,13600,1,1
30,1,1,1,1,0,1,1,1,1,377,387,21100,0,1
56,1,1,1,1,0,1,1,1,0,390,?,14100,0,1 36,1,0,0,1
,0,1,0,1,1,0,372,382,11300,0,1
36,2,0,0,1,0,0,0,1,1,1,370,379,15300,0,1 33,1,0
,0,1,0,1,0,1,1,0,367,376,17400,0,1
19,1,0,0,1,0,0,0,1,1,0,361,375,17600,0,1 12
,1,0,0,1,0,1,0,1,1,0,364,370,12900,0,0
...
```

Ohne näher auf die Datenbank einzugehen, ist es wichtig zu erwähnen, dass nur Patienten in die Datenbank aufgenommen werden, bei denen bei der Ankunft im Krankenhaus der Verdacht auf eine Blinddarmentzündung bestand und die dann operiert wurden. In der siebten Zeile sehen wir, dass C4.5 auch mit fehlenden Werten umgehen kann. Die Daten umfassen 9764 Fä

```
unixprompt> c4.5 -f app -u -m 100
```

C4.5 [Release 8] Entscheidungsbaumgenerator

Mi, 23. August 13:13:15 2006

Lesen Sie 9764 Fälle (15 Attribute) aus `app.data`

Entscheidungsbaum:

Leukozyten <= 11030 :

| Rebound_tenderness = 0:

|| Temp_rektal > 381 : 1 (135,9/54,2)

|| Temp_rektal <= 381 :

||| Local_guarding = 0: 0 (1453,3/358,9)

||| Local_guarding = 1:

|||| Sex_(1=m___2=w) = 1: 1 (160,1/74,9)

|||| Sex_(1=m___2=w) = 2: 0 (286,3/97,6)

| Rebound_tenderness = 1:

|| Leukozyten <= 8600 :

|| Temp_rektal > 378 : 1 (176,0/59,4)

|| Temp_rektal <= 378 :

|||| Sex_(1=m___2=w) = 1:

|||| Local_guarding = 0: 0 (110,7/51,7)

|||| Local_guarding = 1: 1 (160,6/68,5)

|||| Sex_(1=m___2=w) = 2:

|||| Alter <= 14: 1 (131,1/63,1)

|||| Alter > 14: 0 (398,3/137,6)

|| Leukozyten > 8600:

|| Sex_(1=m___2=w) = 1: 1 (429,9/91,0)

|| Sex_(1=m___2=w) = 2:

|||| Local_guarding = 1: 1 (311,2/103,0)

|||| Local_guarding = 0:

|||| Temp_rektal <= 375 : 1 (125,4/55,8)

|||| Temp_rektal > 375 : 0 (118,3/56,1)

Leukozyten > 11030:

| Rebound_tenderness = 1: 1 (4300,0/519,9)

| Rebound_tenderness = 0:

|| Leukozyten > 14040 : 1 (826,6/163,8)

|| Leukozyten <= 14040 :

|| Pain_on_tapping = 1: 1 (260,6/83,7)

|| Pain_on_tapping = 0:

||| Local_guarding = 1: 1 (117,5/44,4)

||| Local_guarding = 0:

|||| Temp_axial <= 368 : 0 (131,9/57,4)

|||| Temp_axial > 368 : 1 (130,5/57,8)

Vereinfachter Entscheidungsbaum:

Leukozyten > 11030 : 1 (5767,0/964,1)

Leukozyten <= 11030 :

| Rebound_tenderness = 0:

|| Temp_rektal > 381 : 1 (135,9/58,7)

|| Temp_rektal <= 381 :

||| Local_guarding = 0: 0 (1453,3/370,9)

||| Local_guarding = 1:

|||| Sex_(1=m___2=w) = 1: 1 (160,1/79,7)

|||| Sex_(1=m___2=w) = 2: 0 (286,3/103,7)

| Rebound_tenderness = 1:

|| Leukozyten > 8600 : 1 (984,7/322,6)

|| Leukozyten <= 8600 :

```

      | Temp_rektal > 378 : 1 (176,0/64,3)
      | Temp_rectal <= 378 :
      |   | Sex_(1=m_2=w) = 1:
      |   |   | Local_guarding = 0: 0 (110,7/55,8)
      |   |   | Local_guarding = 1: 1 (160,6/73,4)
      |   | Sex_(1=m_2=w) = 2:
      |   |   | Alter <= 14: 1 (131,1/67,6)
      |   |   | Alter > 14: 0 (398,3/144,7)

```

Auswertung der Trainingsdaten (9764 Items):

Vor dem Beschneiden Nach dem Beschneiden

Größenfehler Schätzung der Größenfehler

37 2197(22,5%) 21 2223(22,8%) (23,6%) <<

Auswertung anhand von Testdaten (4882 Items):

Vor dem Beschneiden Nach dem Beschneiden

Größenfehler Schätzung der Größenfehler

37 1148(23,5%) 21 1153(23,6%) (23,6%) <<

(a) (b) <-klassifiziert als

758 885 (a): Klasse 0

268 2971 (b): Klasse 1

8.4.6 Kontinuierliche Attribute

In den für die Blinddarmentzündungsdiagnose generierten Bäumen gibt es einen Knoten Leukozyten > 11030, was eindeutig auf das kontinuierliche Attribut Leukozyten zurückzuführen ist durch Festlegen eines Schwellenwerts auf den Wert 11030. C4.5 hat somit aus dem kontinuierlichen Attribut Leukozyten ein binäres Attribut Leukozyten > 11030 erstellt.

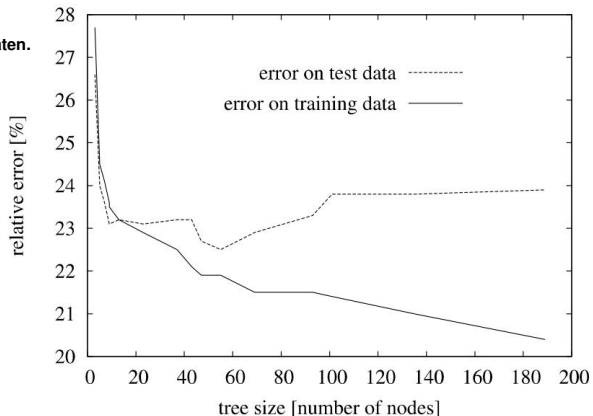
Der Schwellenwert $\hat{y}_{D,A}$ für ein Attribut A wird durch den folgenden Algorithmus bestimmt:

Für alle Werte v, die in den Trainingsdaten D vorkommen, gilt das binäre Attribut $A > v$ generiert und deren Informationsgewinn berechnet. Der Schwellenwert $\hat{y}_{D,A}$ wird dann auf gesetzt der Wert v mit dem maximalen Informationsgewinn, also:

$$\hat{y}_{D,A} = \operatorname{argmax}_v \{G(D, A > v)\}.$$

Für ein Attribut wie den Leukozytenwert oder das Alter des Patienten kann eine Entscheidung getroffen werden eine binäre Diskretisierung ist vermutlich zu ungenau. Dennoch besteht keine Notwendigkeit Diskretisieren Sie feiner, da jedes kontinuierliche Attribut bei jedem neu generierten Attribut getestet wird Knoten und kann daher in einem Baum mit unterschiedlichem Schwellenwert $\hat{y}_{D,A}$ wiederholt vorkommen. Daher letztlich erhalten wir eine sehr gute Diskretisierung, deren Feinheit zum Problem passt.

Abb. 8.27 Lernkurve von C4.5 anhand der Blinddarmentzündungsdaten. Wir sehen deutlich die Überanpassung von Bäumen mit mehr als 55 Knoten



8.4.7 Beschneiden – Den Baum fällen

Seit Aristoteles gilt die Regel, dass von zwei wissenschaftlichen Theorien, die den gleichen Sachverhalt gleich gut erklären, die einfachere zu bevorzugen ist. Dieses Gesetz der Ökonomie, heute auch als *Occams Rasiermesser bekannt*, ist für maschinelles Lernen und Data Mining von großer Bedeutung.

Ein Entscheidungsbaum ist eine Theorie zur Beschreibung der Trainingsdaten. Eine andere Theorie zur Beschreibung der Daten sind die Daten selbst. Wenn der Baum alle Daten fehlerfrei klassifiziert, aber viel kompakter und damit für den Menschen leichter verständlich ist, dann ist er nach Occams Rasiermesser vorzuziehen. Das Gleiche gilt für zwei Entscheidungsbäume unterschiedlicher Größe. Daher muss das Ziel jedes Algorithmus zur Generierung eines Entscheidungsbaums darin bestehen, für eine gegebene Fehlerrate einen möglichst kleinen Entscheidungsbaum zu generieren. Unter allen Bäumen mit einer festen Fehlerrate sollte immer der kleinste Baum

Bisher haben wir den Begriff Fehlerquote nicht genau definiert. Wie bereits mehrfach erwähnt, ist es wichtig, dass der Lernbaum die Trainingsdaten nicht nur im Gedächtnis behält, sondern dass diese gut verallgemeinert werden können. Um die Generalisierungsfähigkeit eines Baums zu testen, teilen wir die verfügbaren Daten in einen Satz *Trainingsdaten* und einen Satz *Testdaten* auf. Die Testdaten werden vor dem Lernalgorithmus verborgen und nur zum Testen verwendet. Wenn ein großer Datensatz verfügbar ist, beispielsweise die Blinddarmentzündungsdaten, können wir beispielsweise zwei Drittel der Daten zum Lernen und das verbleibende Drittel zum Testen verwenden.

Neben der besseren Verständlichkeit hat Occams Rasiermesser noch eine weitere wichtige Begründung: die Fähigkeit zur Generalisierung. Je komplexer das Modell (hier ein Entscheidungsbaum) ist, desto mehr Details werden abgebildet, aber im gleichen Maße ist das Modell umso weniger auf neue Daten übertragbar. Dieser Zusammenhang ist in Abb. 8.27 dargestellt. Entscheidungsbäume unterschiedlicher Größe wurden anhand der Blinddarmentzündungsdaten trainiert. In der Grafik sind Klassifizierungsfehler sowohl der Trainingsdaten als auch der Testdaten angegeben. Die Fehlerrate der Trainingsdaten nimmt monoton mit der Größe des Baums ab. Bis zu einer Baumgröße von 55 Knoten sinkt auch die Fehlerquote bei Testdaten. Wächst der Baum jedoch weiter, steigt die Fehlerquote wieder an! Dieser Effekt, den wir bereits bei der Methode des nächsten Nachbarn gesehen haben, wird als *Überanpassung* bezeichnet.

Wir werden dieses Konzept, das für fast alle Lernprozesse wichtig ist, weitergeben:
eine allgemeine Definition aus [Mit97]:

Definition 8.7 Es sei ein spezifischer Lernalgorithmus, also ein Lernagent, gegeben. Wir nennen einen Agenten A eine Überanpassung an die Trainingsdaten, wenn es einen anderen Agenten A' gibt, dessen Fehler in den Trainingsdaten größer als der von A , dessen Fehler in der gesamten Datenverteilung jedoch kleiner als der Fehler von A ist.

Wie können wir nun diesen Punkt des minimalen Fehlers in den Testdaten finden? Der offensichtlichste Algorithmus heißt *Kreuzvalidierung*. Während der Baumkonstruktion wird parallel der Fehler der Testdaten gemessen. Sobald der Fehler deutlich ansteigt, wird der Baum mit dem minimalen Fehler gespeichert. Dieser Algorithmus wird vom zuvor erwähnten CART-System verwendet.

C4.5 funktioniert etwas anders. Zunächst wird mit dem Algorithmus **GENERATEDECISIONTREE** aus Abb. 8.26 auf Seite 193 ein Baum generiert, der in der Regel überfit ist. Mithilfe *von Pruning* wird dann versucht, Knoten des Baums wegzuschneiden, bis der Fehler in den Testdaten, der durch den Fehler in den Trainingsdaten geschätzt wird, zuzunehmen beginnt.⁸ Wie die Konstruktion des Baums ist auch dies ein gieriger Algorithmus. Dies bedeutet, dass ein einmal beschnittener Knoten nicht erneut eingefügt werden kann, auch wenn sich dies später als besser herausstellt.

8.4.8 Fehlende Werte

Häufig fehlen einzelne Attributwerte in den Trainingsdaten. Im LEX MED -Datensatz kommt der folgende Eintrag vor:

56, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14100, 0, 1,

bei dem einer der Fieberwerte fehlt. Dennoch können solche Daten beim Aufbau des Entscheidungsbaums verwendet werden. Wir können dem Attribut den häufigsten Wert aus dem gesamten Datensatz oder den häufigsten aller Datenpunkte derselben Klasse zuweisen. Noch besser ist es, den fehlenden Attributwert durch die Wahrscheinlichkeitsverteilung aller Attributwerte zu ersetzen und das Trainingsbeispiel entsprechend dieser Verteilung in Zweige aufzuteilen. Dies ist übrigens ein Grund für das Auftreten nicht ganzzahliger Werte in den Ausdrücken in Klammern neben den Blattknoten des C4.5-Baums.

Fehlende Werte können nicht nur beim Lernen, sondern auch beim Klassifizieren auftreten. Diese werden auf die gleiche Weise gehandhabt wie beim Lernen.

⁸ Es wäre besser, den Fehler direkt auf die Testdaten anzuwenden. Zumindest dann, wenn die Menge an Trainingsdaten ausreicht, um einen separaten Testsatz zu rechtfertigen.

8.4.9 Zusammenfassung

Das Erlernen von Entscheidungsbäumen ist ein beliebter Ansatz für Klassifizierungsaufgaben. Die Gründe hierfür liegen in der einfachen Anwendung und Schnelligkeit. Bei einem Datensatz von etwa 10.000 LEXMED- Datenpunkten mit jeweils 15 Attributen benötigt C4.5 etwa 0,3 Sekunden zum Lernen. Dies ist im Vergleich zu anderen Lernalgorithmen sehr schnell.

Für den Anwender ist es aber auch wichtig, dass der Entscheidungsbaum als erlerntes Modell verstanden und ggf. verändert werden kann. Es ist auch nicht schwierig, einen Entscheidungsbaum automatisch in eine Reihe von If-Then-Else-Anweisungen umzuwandeln und ihn so effizient in ein bestehendes Programm einzubauen.

Da sowohl beim Aufbau des Baums als auch beim Beschneiden ein Greedy-Algorithmus verwendet wird, sind die Bäume im Allgemeinen suboptimal. Der entdeckte Entscheidungsbaum weist normalerweise eine relativ geringe Fehlerquote auf. Es gibt jedoch möglicherweise einen besseren Baum, da die heuristische gierige Suche von C4.5 kleine Bäume und Attribute mit hohem Informationsgewinn an der Spitze des Baums bevorzugt. Bei Attributen mit vielen Werten weist die vorgestellte Formel für den Informationsgewinn Schwächen auf. Alternativen hierzu werden in [Mit97] angegeben.

8.5 Lernen von Bayes'schen Netzwerken

In Abschn. In Abschnitt 7.4 wurde gezeigt, wie man manuell ein Bayes'sches Netzwerk aufbaut. Jetzt werden wir Algorithmen zur Induktion von Bayes'schen Netzwerken vorstellen. Ähnlich wie beim zuvor beschriebenen Lernprozess wird ein Bayes-Netzwerk automatisch aus einer Datei mit Trainingsdaten generiert. Dieser Prozess gliedert sich typischerweise in zwei Teile.

1. *Erlernen der Netzwerkstruktur:* Für gegebene Variablen wird aus den Trainingsdaten die Netzwerktopologie generiert. Dieser erste Schritt ist bei weitem der schwierigere und wird später genauer beachtet.
2. *Lernen der bedingten Wahrscheinlichkeiten:* Für bekannte Netzwerktopologien müssen die CPTs mit Werten gefüllt werden. Wenn genügend Trainingsdaten verfügbar sind, können alle notwendigen bedingten Wahrscheinlichkeiten durch Zählen der Häufigkeiten in den Daten geschätzt werden. Dieser Schritt lässt sich relativ einfach automatisieren.

Wir erklären nun, wie Bayesianische Netzwerke mithilfe eines einfachen Algorithmus aus [Jen01] lernen.

8.5.1 Erlernen der Netzwerkstruktur

Bei der Entwicklung eines Bayes'schen Netzwerks (siehe Abschn. 7.4.6) muss die kausale Abhängigkeit der Variablen berücksichtigt werden, um ein einfaches Netzwerk guter Qualität zu erhalten. Der menschliche Entwickler ist auf Hintergrundwissen angewiesen, das der Maschine nicht zur Verfügung steht. Daher kann dieser Vorgang nicht einfach automatisiert werden.

Das Finden einer optimalen Struktur für ein Bayes'sches Netzwerk kann als klassisches Suchproblem formuliert werden. Gegeben seien ein Satz Variablen $V_1, \dots, V_n$ und eine Datei mit Trainingsdaten. Wir suchen nach einer Menge gerichteter Kanten ohne Kreise zwischen den Knoten

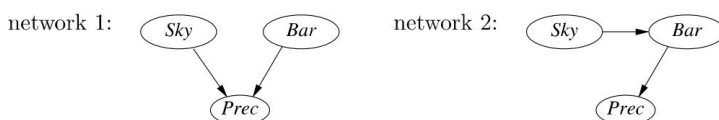


Abb. 8.28 Zwei Bayes'sche Netze zur Modellierung des Wettervorhersagebeispiels aus Übung 7.3 auf Seite 158

$V_1, \dots, V_n$, also ein gerichteter azyklischer Graph (DAG), der die zugrunde liegenden Daten bestmöglich wiedergibt.

Zuerst beobachten wir den Suchraum. Die Anzahl verschiedener DAGs wächst mehr als exponentiell mit der Anzahl der Knoten. Für fünf Knoten gibt es 29281 und für neun Knoten etwa 1015 verschiedene DAGs [MDBM00]. Somit ist eine uninformierte kombinatorische Suche (siehe Abschn. 6.2) im Raum aller Graphen mit einer gegebenen Menge von Variablen aussichtslos, wenn die Anzahl der Variablen wächst. Daher müssen heuristische Algorithmen verwendet werden. Dies wirft die Frage nach einer Bewertungsfunktion für Bayes'sche Netzwerke auf. Es ist möglich, den Klassifizierungsfehler eines Netzwerks bei der Anwendung auf einen Testdatensatz zu messen, wie es beispielsweise in C4.5 durchgeführt wird (siehe Abschn. 8.4). Hierzu müssen allerdings die vom Bayes'schen Netzwerk berechneten Wahrscheinlichkeiten

Über die Wahrscheinlichkeitsverteilung kann eine direkte Messung der Qualität eines Netzwerks erfolgen. Wir gehen davon aus, dass wir vor dem Aufbau des Netzwerks aus den Daten die Verteilung bestimmen (schätzen) konnten. Dann beginnen wir mit der Suche im Raum aller DAGs, schätzen anhand der Daten den Wert der CPTs für jede DAG (also für jedes Bayes'sche Netzwerk), berechnen daraus die Verteilung und vergleichen sie mit der bekannten Verteilung der Daten. Für den Vergleich von Verteilungen benötigen wir natürlich eine Distanzmetrik.

Betrachten wir das Wettervorhersagebeispiel aus Übung 7.3 auf Seite 158 mit den drei Variablen *Sky*, *Bar*, *Prec* und der Verteilung

$$P(\text{Sky}, \text{Bar}, \text{Prec}) = (0,40, 0,07, 0,08, 0,10, 0,09, 0,11, 0,03, 0,12).$$

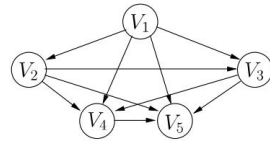
In Abb. 8.28 sind zwei Bayes-Netzwerke dargestellt, die wir nun hinsichtlich ihrer Qualität vergleichen werden. Jedes dieser Netzwerke geht von einer Unabhängigkeitsannahme aus, die bestätigt wird, indem wir die Verteilung des Netzwerks bestimmen und diese dann mit der ursprünglichen Verteilung vergleichen (siehe Übung 8.15 auf Seite 219).

Da die Verteilung für konstante vorgegebene Variablen eindeutig durch einen Vektor konstanter Länge dargestellt wird, können wir die euklidische Norm der Differenz der beiden Vektoren als Abstand zwischen Verteilungen berechnen. Wir definieren

$$dq(x, y) = \sum (x_i - y_i)^2$$

als Summe der Quadrate der Abstände der Vektorkomponenten und berechnen Sie den Abstand $dq(P_a, P) = 0,0029$ der Verteilung P_a von Netzwerk 1 von der ursprünglichen Verteilung. Für Netzwerk 2 berechnen wir $dq(P_b, P) = 0,014$. Offensichtlich ist Netzwerk 1 eine bessere Annäherung an die Verteilung. Oftmals wird anstelle des Quadratabstandes der

Abb. 8.29 Das maximale Netzwerk mit fünf Variablen und Kanten (V_i, V_j) , die die Bedingung $i < j$ erfüllen



sogenannte Kullback-Leibler-Distanz

$$dk(x, y) = - \sum_i y_i (\log_2 y_i - \log_2 x_i),$$

Es wird eine Metrik der Informationstheorie verwendet. Damit berechnen wir $dk(P_a, P) = 0,017$ und $dk(P_b, P) = 0,09$ und kommen zum gleichen Ergebnis wie zuvor. Es ist zu erwarten, dass Netzwerke mit vielen Kanten die Verteilung besser annähern als solche mit wenigen Kanten. Wenn alle Kanten im Netzwerk konstruiert werden, wird es sehr unübersichtlich und es besteht die Gefahr einer Überanpassung, wie es bei vielen anderen Lernalgorithmen der Fall ist. Um eine Überanpassung zu vermeiden, geben wir kleinen Netzwerken mithilfe einer heuristischen Bewertungsfunktion ein größeres Gewicht

$$f(N) = \text{Größe}(N) + w \cdot dk(P_N, P).$$

Dabei ist $\text{Size}(N)$ die Anzahl der Einträge in den CPTs und P_N die Verteilung des Netzwerks N . w ist ein Gewichtungsfaktor, der manuell angepasst werden muss.

Der Lernalgorithmus für Bayes'sche Netzwerke berechnet also die heuristische Bewertung $f(N)$ für viele verschiedene Netzwerke und wählt dann das Netzwerk mit dem kleinsten Wert aus. Wie bereits erwähnt besteht die Schwierigkeit in der Reduzierung des Suchraums für die gesuchte Netzwerktopologie. Als einfacher Algorithmus ist es möglich, ausgehend von einer (z. B. kausalen) Ordnung der Variablen $V_1, \dots, V_n$, nur solche Kanten in den Graphen aufzunehmen, für die $i < j$ gilt. Wir beginnen mit dem Maximalmodell, das diese Bedingung erfüllt. Dieses Netzwerk ist in Abb. 8.29 für fünf geordnete Variablen dargestellt.

Nun wird beispielsweise im Sinne einer Greedy Search (vgl. Abschn. 6.3.1) eine Kante nach der anderen entfernt, bis der Wert f nicht mehr abnimmt.

Für größere Netzwerke ist dieser Algorithmus in dieser Form nicht praktikabel. Der große Suchraum, die manuelle Abstimmung des Gewichts w und der notwendige Vergleich mit einer Zielverteilung P sind Gründe dafür, denn diese können einfach zu groß werden, oder der verfügbare Datensatz könnte zu klein sein.

Tatsächlich ist die Forschung zum Lernen von Bayes'schen Netzwerken immer noch in vollem Gange, und es gibt eine große Anzahl vorgeschlagener Algorithmen, zum Beispiel den EM-Algorithmus (siehe Abschn. 8.7.2), Markov-Ketten-Monte-Carlo-Methoden und Gibbs-Sampling [DHS01, Jor99, Jen01, HTF09]. Neben dem hier vorgestellten Batch-Learning, bei dem das Netzwerk einmalig aus dem gesamten Datensatz generiert wird, gibt es auch inkrementelle Algorithmen, bei denen jeder einzelne neue Fall zur Verbesserung des Netzwerks genutzt wird. Es gibt auch Implementierungen dieser Algorithmen, beispielsweise von Hugin (www.hugin.com) und Bayesware (www.bayesware.com).

8.6 Der Naive-Bayes-Klassifikator

In Abb. 7.14 auf Seite 152 wurde die Diagnose einer Blinddarmentzündung als Bayes'sches Netzwerk modelliert. Da gerichtete Kanten an einem Diagnoseknoten beginnen und keine Kanten dort enden, muss zur Beantwortung einer Diagnoseabfrage die Bayes-Formel verwendet werden. Für die Symptome $S_1, \dots, S_n$ und die k -Wert-Diagnose D mit den Werten $b_1, \dots, b_k$ berechnen wir die Wahr

$$P(D|S_1, \dots, S_n) = P \frac{P(S_1, \dots, S_n|D) \cdot P(D)}{(S_1, \dots, S_n)},$$

für die Diagnose anhand der Symptome des Patienten. Im schlimmsten Fall, also wenn es keine unabhängigen Variablen gäbe, müssten alle Kombinationen aller Symptome und D für alle 20.643.840 Wahrscheinlichkeiten der Verteilung $P(S_1, \dots, S_n, D)$ bestimmt werden. Dies würde eine riesige Datenbank erfordern. Im Fall des Bayes'schen Netzwerks von LEX MED reduziert sich die Anzahl der notwendigen Werte (in den CPTs) auf 521. Das Netzwerk kann jedoch weiter vereinfacht werden, indem wir annehmen, dass alle Symptomvariablen bei gegebenem D bedingt unabhängig sind, das heißt:

$$P(S_1, \dots, S_n|D) = P(S_1|D) \cdots P(S_n|D).$$

Das Bayes'sche Netzwerk für Blinddarmentzündung wird dann zu dem in Abb. 8.30 auf Seite 203 gezeigten Stern vereinfacht. Somit erhalten wir die Formel

$$S_1, \dots, S_n) = \frac{P(D) \prod_{i=1}^n P(S_i|D)}{P(S_1, \dots, S_n)}. \quad (8.8)$$

Die berechneten Wahrscheinlichkeiten werden durch einen einfachen naiven Bayes-Klassifikator in eine Entscheidung umgewandelt, der aus allen Werten d_i in D das Maximum $P(D = d_i | S_1, \dots, S_n)$ auswählt. Das heißt, es bestimmt

$$d_{\text{Naive-Bayes}} = \operatorname{argmax}_{i \in \{1, \dots, k\}} P(D = d_i | S_1, \dots, S_n).$$

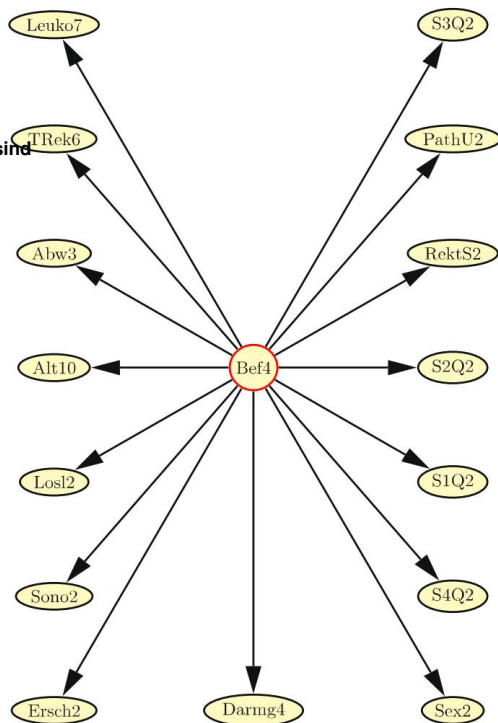
Da der Nenner in (8.8) konstant ist, kann er bei der Maximierung weggelassen werden, was zur *naiven Bayes-Formel* führt

$$d_{\text{Naive-Bayes}} = \operatorname{argmax}_{i \in \{1, \dots, k\}} P(D = d_i) \prod_{j=1}^n P(S_j | d_i).$$

Da mehrere Knoten nun weniger Vorfahren haben, verringert sich die Anzahl der zur Beschreibung der LECMED- Verteilung in den CPTs notwendigen Werte gemäß (7.22) auf Seite 151 auf

$$6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 4 + 3 \cdot 4 + 10 \cdot (1 \cdot 4) + 1 = 141.$$

Abb. 8.30 Bayesianisches Netzwerk für die LEXMED-Anwendung mit der Annahme, dass alle Symptome bei gegebener Diagnose bedingt unabhängig sind



Für ein medizinisches Diagnosesystem wie LEXYMED wäre diese Vereinfachung nicht akzeptabel. Aber für Aufgaben mit vielen unabhängigen Variablen ist Naive Bayes teilweise oder sogar sehr gut geeignet, wie wir am Beispiel der Textklassifizierung sehen werden.

Die naive Bayes-Klassifikation ist übrigens genauso aussagekräftig wie das in Abschn. 2.1 beschriebene lineare Score-System. 7.3.1 (siehe Übung 8.16 auf Seite 219). Das heißt, allen Scores liegt die Annahme zugrunde, dass alle Symptome angesichts der Diagnose bedingt unabhängig sind. Dennoch werden Scores auch heute noch in der Medizin verwendet. Trotz der Tatsache, dass er aus einer besseren, repräsentativen Datenbasis generiert wurde, weist der Ohmann-Score im Vergleich zu LEXMED in Abb. 7.10 auf Seite 142 eine schlechtere Diagnosequalität auf. Die eingeschränkte Aussagekraft ist sicherlich ein Grund dafür. Beispielsweise ist es wie bei Naive Bayes nicht möglich, Abhängigkeiten zwischen Symptomen zu berücksichtigen.

Schätzung von Wahrscheinlichkeiten Wenn wir die naive Bayes-Formel in (8.8) auf Seite 202 betrachten, sehen wir, dass der gesamte Ausdruck Null wird, sobald einer der Faktoren $P(S_i | D)$ auf der rechten Seite Null wird. Theoretisch ist hier nichts falsch. In der Praxis kann dies jedoch zu sehr unangenehmen Effekten führen, wenn die $P(S_i | D)$ klein sind, da diese durch Zählen von Frequenzen und deren Substitution geschätzt werden

hinein

$$P(S_i = x | D = y) = \frac{|S_i = x \cap D = y|}{|D = y|}.$$

Nehmen Sie an, dass für die Variablen S_i : $P(S_i = x | D = y) = 0,01$ und dass es 40 Trainingsfälle mit $D = y$ gibt. Dann gibt es mit hoher Wahrscheinlichkeit keinen Trainingsfall mit $S_i = x$ und $D = y$, und wir schätzen $P(S_i = x | D = y) = 0$. Für einen anderen Wert $D = z$ nehmen wir an, dass die Beziehungen ähnlich liegen, aber die Schätzung ergibt Werte größer als Null für alle $P(S_i = x | D = z)$. Daher wird immer der Wert $D = z$ bevorzugt, der nicht die tatsächliche Wahrscheinlichkeitsverteilung widerspiegelt. Daher gilt bei der Schätzung von Wahrscheinlichkeiten die Formel

$$P(A|B) \approx \frac{|A \cap B|}{|B|} = \frac{\text{schnappen}}{nB}$$

wird ersetzt durch

$$P(A|B) \approx \frac{nAB + mp}{nB + m},$$

wobei $p = P(A)$ die A-priori-Wahrscheinlichkeit für A ist und m eine frei wählbare Konstante ist, die als „äquivalente Datengröße“ bekannt ist. Je größer m wird, desto größer ist das Gewicht der A-priori-Wahrscheinlichkeit im Vergleich zu dem aus der gemessenen Häufigkeit ermittelten Wert.

8.6.1 Textklassifizierung mit Naive Bayes

Naive Bayes ist heute sehr erfolgreich und produktiv in der Textklassifizierung. Seine primäre und zugleich sehr wichtige Anwendung ist die automatische Filterung von E-Mails in gewünschte und unerwünschte bzw. Spam-E-Mails. In Spam-Filtern wie SpamAssassin [Sch04] kommt unter anderem ein naiver Bayes-Klassifikator zum Einsatz, der lernt, gewünschte E-Mails von Spam zu trennen. SpamAssassin ist ein Hybridsystem, das eine anfängliche Filterung mithilfe von Black- und Whitelists durchführt. Schwarze Listen sind Listen blockierter E-Mail-Adressen von Spam-Absendern, deren E-Mails immer gelöscht werden, und weiße Listen sind Listen mit Absendern, deren E-Mails immer zugestellt werden. Nach dieser Vorfilterung werden die verbleibenden E-Mails durch den naiven Bayes-Klassifikator nach ihrem eigentlichen Inhalt, also nach dem Text, klassifiziert. Der erkannte Klassenwert wird dann zusammen mit anderen Attributen aus dem Header der E-Mail wie der Domäne des Absenders, dem MIME-Typ usw. anhand einer

Hier ist die Lernfähigkeit des naiven Bayes-Filters sehr wichtig. Hierzu muss der Benutzer zunächst eine große Anzahl an E-Mails manuell als Wunsch- oder Spam-E-Mails klassifizieren.

Anschließend wird der Filter trainiert. Um auf dem neuesten Stand zu bleiben, muss der Filter regelmäßig neu trainiert werden.

Hierzu sollte der Benutzer alle E-Mails, die vom Filter fälschlicherweise klassifiziert wurden, korrekt klassifizieren, also in die entsprechenden Ordner ablegen. Der Filter wird dann mit diesen E-Mails kontinuierlich neu trainiert.

Neben der Spam-Filterung gibt es viele weitere Anwendungen zur automatischen Textklassifizierung. Wichtige Anwendungen sind das Filtern unerwünschter Einträge in Internet-Diskussionsforen und das Tracking von Websites mit kriminellen Inhalten wie militanten oder terroristischen Aktivitäten, Kinderpornografie oder Rassismus. Darüber hinaus lassen sich damit auch Suchmaschinen an die Präferenzen des Nutzers anpassen, um die Suchergebnisse besser einzuordnen. Im industriellen und wissenschaftlichen Umfeld steht die unternehmensweite Suche in Datenbanken oder in der Literatur im Vordergrund der Forschung. Durch seine Lernfähigkeit kann sich ein Filter an die Gewohnheiten und Wünsche anpassen.

Wir werden die Anwendung von Naive Bayes auf die Textanalyse anhand eines kurzen Beispiels vorstellen ausführlicher Text von Alan Turing aus [Tur50]:

„Wir können hoffen, dass Maschinen irgendwann in allen rein intellektuellen Bereichen mit Menschen konkurrieren werden. Aber welche sind die besten für den Anfang? Auch das ist eine schwierige Entscheidung. Viele Leute denken, dass eine sehr abstrakte Aktivität, wie das Schachspielen, am besten wäre. Man kann auch behaupten, dass es am besten ist, die Maschine mit den besten Sinnesorganen auszustatten, die man für Geld kaufen kann, und ihr dann beizubringen, Englisch zu verstehen und zu sprechen. Dieser Prozess könnte dem normalen Unterricht eines Kindes folgen. Dinge würden hervorgehoben und benannt usw. Auch hier weiß ich nicht, was die richtige Antwort ist, aber ich denke, dass b

Nehmen wir an, dass Texte wie dieser in zwei Klassen eingeteilt werden sollten: „I“ für „interessant“ und „-I“ für „uninteressant“. Nehmen wir außerdem an, dass eine Datenbank mit Texten existiert, die bereits klassifiziert sind. Welche Attribute sollen verwendet werden? Bei einem klassischen Ansatz zum Aufbau eines Bayes'schen Netzwerks definieren wir eine Reihe von Attributen wie die Länge des Textes, die durchschnittliche Satzlänge, die relative Häufigkeit bestimmter Satzzeichen, die Häufigkeit mehrerer wichtiger Wörter wie „I“, „Maschinen“ usw. Bei der Klassifizierung mittels Naive Bayes wird dagegen ein überraschend primitiver Algorithmus gewählt. Für jede der n Wortpositionen im Text wird ein Attribut s_i definiert. Als mögliche Werte für alle Positionen s_i sind alle im Text vorkommenden Wörter erlaubt. Nun für die Klassen I und -I die Werte

$$P(I | s_1, \dots, s_n) = c \cdot \prod_{i=1}^n P(s_i | I) \quad (8.9)$$

und $P(-I | s_1, \dots, s_n)$ berechnet werden und dann die Klasse mit dem Maximalwert ausgewählt werden. Im obigen Beispiel mit insgesamt 113 Wörtern ergibt dies

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \cdot P(s_1 = \text{„Wir“} | I) \cdot P(s_2 = \text{„kann“} | I) \cdot \dots \cdot P(s_{113} = \text{„sollte“} | I)$$

Und

$$\begin{aligned} P(-I | s_1, \dots, s_n) &= c \cdot P(-I) \cdot P(s_1 = \text{„Wir“} | -I) \\ &\quad \cdot P(s_2 = \text{„kann“} | -I) \cdot \dots \cdot P(s_{113} = \text{„sollte“} | -I). \end{aligned}$$

Das Lernen hier ist ganz einfach. Die bedingten Wahrscheinlichkeiten $P(s_i | I)$, $P(s_i | -I)$ und die A-priori-Wahrscheinlichkeiten $P(I)$, $P(-I)$ müssen einfach berechnet werden. Wir gehen nun zusätzlich davon aus, dass die $P(s_i | I)$ unabhängig von der Position im Text sind. Das

bedeutet zum Beispiel das

$$P(s_{61} = \text{„und“} | I) = P(s_{69} = \text{„und“} | I) = P(s_{86} = \text{„und“} | I).$$

Wir könnten also den Ausdruck $P(\text{und} | I)$ mit der neuen binären Variablen *und* als Wahrscheinlichkeit des Auftretens von „und“ an einer beliebigen Position verwenden.

Die Umsetzung kann etwas beschleunigt werden, wenn wir die Häufigkeit n_i von finden jedes Wort, das im Text vorkommt, und verwenden Sie die Formel

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \prod_{i=1}^I P(w_i | I) n_i \quad (8.10)$$

was äquivalent zu (8.9) auf Seite 205 ist. Bitte beachten Sie, dass der Index i im Produkt nur auf die Anzahl I verschiedener Wörter verweist, die im Text vorkommen.

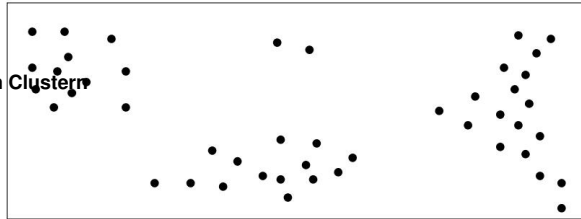
Trotz seiner Einfachheit liefert Naive Bayes hervorragende Ergebnisse für die Textklassifizierung. Spamfilter, die mit Naive Bayes arbeiten, erreichen Fehlerraten von deutlich unter einem Prozent. Die Systeme DSPAM und CRM114 können sogar so gut trainiert werden, dass sie nur eine von 7000 bzw. 8000 E-Mails falsch klassifizieren. Dies entspricht einer Richtigkeit von nahezu 99,99 %.

8.7 Clustering

Wenn wir in einer Suchmaschine nach dem Begriff „Mars“ suchen, erhalten wir Ergebnisse wie „der Planet Mars“ und „Schokoladen-, Süßwaren- und Getränkekonglomerat“, die semantisch recht unterschiedlich sind. In der Menge der entdeckten Dokumente gibt es zwei deutlich unterschiedliche *Cluster*. Google beispielsweise listet die Ergebnisse immer noch unstrukturiert auf. Es wäre besser, wenn die Suchmaschine die Cluster trennen und dem Benutzer entsprechend präsentieren würde, da der Benutzer normalerweise nur an einem der Cluster interessiert ist.

Der Unterschied zwischen *Clustering* und überwachtem Lernen besteht darin, dass die Trainingsdaten unbeschriftet sind. Somit fehlt die Vorstrukturierung der Daten durch den Betreuer. Vielmehr geht es beim Clustering darum, Strukturen zu finden. Im Bereich der Trainingsdaten finden sich Datenansammlungen wie in Abb. 8.31 auf Seite 207. In einem Cluster ist der Abstand benachbarter Punkte typischerweise kleiner als der Abstand zwischen Punkten verschiedener Cluster. Daher ist die Wahl einer geeigneten Distanzmetrik für Punkte, also für zu gruppierende Objekte und für Cluster, von grundlegender Bedeutung. Wie zuvor gehen wir im Folgenden davon aus, dass jedes Datenobjekt durch einen Vektor numerischer Attribute beschrieben wird.

Abb. 8.31
Einfaches
zweidimensionales
Beispiel mit vier klar getrennten Clustern



8.7.1 Distanzmetriken

Dementsprechend werden für jede Anwendung die verschiedenen Abstandsmetriken für den Abstand d zwischen zwei Vektoren x und y im n -dimensionalen Raum betrachtet. Am häufigsten ist der euklidische Abstand

$$de(x, y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2}.$$

Etwas einfacher ist die Summe quadratierter Abstände

$$dq(x, y) = \sum_{i=1}^N (x_i - y_i)^2,$$

was für Algorithmen, in denen nur Abstände verglichen werden, dem euklidischen Abstand entspricht (Übung 8.19 auf Seite 220). Ebenfalls verwendet wird die bereits erwähnte Manhattan-Distanz

$$dm(x, y) = \sum_{i=1}^N |x_i - y_i|$$

sowie der Abstand der maximalen Komponente

$$d\ddot{y}(x, y) = \max_{i=1, \dots, n} |x_i - y_i|$$

was auf der Maximalnorm basiert. Bei der Textklassifizierung wird die normalisierte Projektion der beiden Vektoren aufeinander, also das normalisierte Skalarprodukt, verwendet

$$\frac{|xy|}{|x||y|}$$

wird häufig berechnet, wobei $|x|$ ist die euklidische Norm von x . Denn diese Formel ist eine Metrik für die Ähnlichkeit der beiden Vektoren, als Distanzmetrik die Umkehrung

$$ds(x, y) = \frac{|x||y|}{|xy|}$$

kann verwendet werden, oder „>“ und „<“ können für alle Vergleiche vertauscht werden.

Bei der Suche nach einem Text werden die Attribute $x_1, \dots, x_n$ ähnlich wie bei Naive Bayes

als Komponenten des Vektors x wie folgt berechnet. Für ein Wörterbuch mit 50.000 Wörtern beträgt der V

entspricht der Häufigkeit des i -ten Wörterbuchworts im Text. Da in einem solchen Vektor normalerweise fast alle Komponenten Null sind, sind bei der Berechnung des Skalarprodukts fast alle Terme der Summation Null. Durch die Nutzung dieser Art von Informationen kann die Implementierung erheblich beschleunigt werden (Übung 8.20 auf Seite 220).

8.7.2 k-Means und der EM-Algorithmus

Immer wenn die Anzahl der Cluster bereits im Voraus bekannt ist, kann der *k-Means*-Algorithmus verwendet werden. Wie der Name schon sagt, werden k Cluster durch ihren Durchschnittswert definiert. Zunächst werden die k Clustermittelpunkte $\tilde{y}_1, \dots, \tilde{y}_k$ zufällig oder manuell initialisiert. Anschließend werden die folgenden zwei Schritte wiederholt durchgeführt: • Klassifizierung aller Daten auf ihren nächsten Clustermittelpunkt • Neuberechnung des Clustermittelpunkts. Als Algorithmus ergibt sich folgendes Schema:

K-MEANS($x_1, \dots, x_n, k$)
 initialisiert $\tilde{y}_1, \dots, \tilde{y}_k$ (z. B. zufällig)

Wiederholen Sie die Klassifizierung von
 $x_1, \dots, \_\_\_\_\_\_$

Die Berechnung des Clustermittelpunkts $\tilde{y}$ für die Punkte $x_1, \dots, x_l$ erfolgt durch

$$\tilde{y} = \frac{1}{l} \sum_{i=1}^l x_i.$$

Die Ausführung an einem Beispiel ist in Abb. 8.32 auf Seite 209 für den Fall zweier Klassen dargestellt. Wir sehen, wie sich nach drei Iterationen die zunächst zufällig ausgewählten Klassenzentren stabilisieren. Obwohl dieser Algorithmus keine Konvergenz garantiert, konvergiert er normalerweise sehr schnell. Das bedeutet, dass die Anzahl der Iterationsschritte typischerweise viel kleiner ist als die Anzahl der Datenpunkte. Seine Komplexität beträgt $O(ndkt)$, wobei n die Gesamtzahl der Punkte, d die Dimensionalität des Merkmalsraums und t die Anzahl der Iterationsschritte ist.

In vielen Fällen stellt die Notwendigkeit, die Anzahl der Unterrichtsstunden im Voraus anzugeben, eine unangenehme Einschränkung dar. Deshalb werden wir als nächstes einen Algorithmus vorstellen, der flexibler ist.

Zuvor erwähnen wir jedoch den *EM-Algorithmus*, der eine kontinuierliche Variante von *k-means* ist, da er keine feste Zuordnung der Daten zu Klassen vornimmt, sondern für jeden Punkt die Wahrscheinlichkeit seiner Zugehörigkeit zu den zurückgibt verschiedene

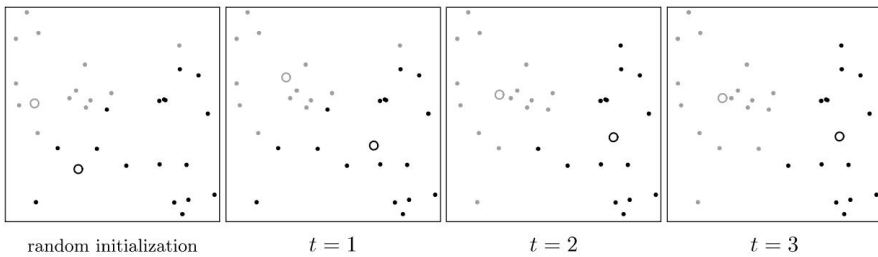


Abb. 8.32 k-Mittelwerte mit zwei Klassen ($k = 2$), angewendet auf 30 Datenpunkte. Ganz links ist der Datensatz mit den anfänglichen Zentren und rechts der Cluster nach jeder Iteration. Nach drei Iterationen ist Konvergenz erreicht

Dabei müssen wir davon ausgehen, dass die Art der Wahrscheinlichkeitsverteilung bekannt ist. Häufig wird die Normalverteilung verwendet. Die Aufgabe des EM-Algorithmus besteht darin, die Parameter (Mittelwert $\bar{y}_i$ und Kovarianzmatrizen $\bar{y}_i$ der k mehrdimensionalen Normalverteilungen) für jeden Cluster zu bestimmen. Ähnlich wie bei k-means werden die beiden folgenden Schritte wiederholt ausgeführt:

Erwartung: Für jeden Datenpunkt wird die Wahrscheinlichkeit $P(C_j | x_i)$ berechnet, dass er zu jedem Cluster gehört.

Maximierung: Anhand der neu berechneten Wahrscheinlichkeiten werden die Parameter der Verteilung neu berechnet.

Dadurch wird eine weichere Clusterbildung erreicht, die in vielen Fällen zu besseren Ergebnissen führt. Dieser Wechsel zwischen Erwartung und Maximierung gibt dem Algorithmus seinen Namen. Neben dem Clustering wird beispielsweise der EM-Algorithmus zum Erlernen von Bayes'schen Netzwerken verwendet [DHS01].

8.7.3 Hierarchisches Clustering

Beim hierarchischen Clustering beginnen wir mit n Clustern, die jeweils aus einem Punkt bestehen. Dann werden die Cluster der nächsten Nachbarn kombiniert, bis alle Punkte zu einem einzigen Cluster zusammengefasst wurden oder bis ein Abbruchkriterium erreicht wurde. Wir erhalten das Schema

HIERARCHICALCLUSTERING($x_1, \dots, x_n, k$) initialisiert
 $C_1 = \{x_1\}, \dots, C_n = \{x_n\}$

Wiederholen. Finden Sie zwei Cluster C_i und C_j mit dem kleinsten Abstand. Kombinieren Sie C_i und C_j , bis die Beendigungsbedingung erreicht ist. Zurückgeben (Baum mit Clustern).

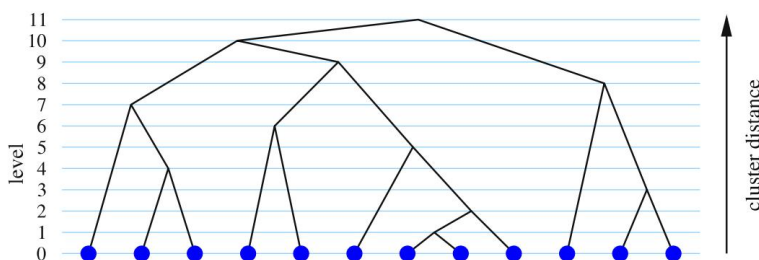


Abb. 8.33 Beim hierarchischen Clustering werden in jedem Schritt die beiden Cluster mit dem kleinsten Abstand zusammengefasst

Als Abbruchbedingung könnte beispielsweise eine gewünschte Anzahl von Clustern oder ein maximaler Abstand zwischen Clustern gewählt werden. In Abb. 8.33 ist dieser Algorithmus schematisch als Binärbaum dargestellt, bei dem von unten nach oben in jedem Schritt, also auf jeder Ebene, zwei Teilbäume verbunden sind. Auf der obersten Ebene werden alle Punkte zu einem großen Cluster zusammengefasst.

Bisher ist unklar, wie die Abstände zwischen den Clustern berechnet werden. Tatsächlich haben wir im vorherigen Abschnitt verschiedene Distanzmetriken für Punkte definiert, diese können jedoch nicht auf Clustern verwendet werden. Eine praktische und häufig verwendete Metrik ist der Abstand zwischen den beiden nächstgelegenen Punkten in den beiden Clustern C_i und C_j :

$$d_{\min}(C_i, C_j) = \min d(x, y) \cdot x \in C_i, y \in C_j$$

Damit erhalten wir den *Nearest-Neighbor-Algorithmus*, dessen Anwendung in Abb. 8.34 auf Seite 211 dargestellt ist. ⁹ Wir sehen, dass dieser Algorithmus eine minimale Spannweite erzeugt. Das Beispiel zeigt außerdem, dass die beiden beschriebenen Algorithmen recht unterschiedliche Cluster erzeugen. Dies zeigt uns, dass bei Diagrammen mit Clustern, die nicht klar getrennt sind, das Ergebnis stark vom Algorithmus oder der gewählten Distanzmetrik abhängt.

Für eine effiziente Implementierung dieses Algorithmus erstellen wir zunächst eine Adjazenzmatrix, in der die Abstände zwischen allen Punkten gespeichert werden, was $O(n^2)$ Zeit und Speicher benötigt. Wenn die Anzahl der Cluster keine Obergrenze hat, wird die Schleife $n - 1$ Mal durchlaufen und die asymptotische Berechnungszeit beträgt $O(n^3)$.

Um den Abstand zwischen zwei Clustern zu berechnen, können wir auch den Abstand $d_{\max}$ verwenden

zwischen den beiden am weitesten entfernten Punkten

$$d_{\max}(C_i, C_j) = \max d(x, y) \cdot x \in C_i, y \in C_j$$

⁹Der Nearest-Neighbor-Algorithmus ist nicht mit der Nearest-Neighbor-Methode für Klassen zu verwechseln. Sifizierung aus Abschn. 8.3.

¹⁰Ein minimaler Spannbaum ist ein azyklischer, ungerichteter Graph mit der minimalen Summe der Kantenlängen.

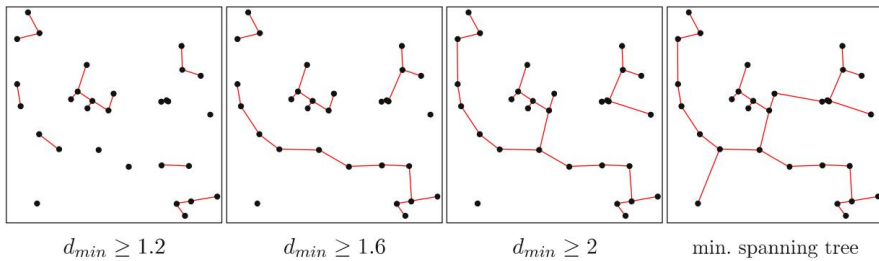


Abb. 8.34 Der Algorithmus für den nächsten Nachbarn, angewendet auf die Daten aus Abb. 8.32 auf Seite 209 auf verschiedenen Ebenen mit 12, 6, 3, 1 Clustern

und erhalten Sie den *Algorithmus für den am weitesten entfernten Nachbarn*. Alternativ wird der Abstand des Clustermittelpunkts $d(\bar{C}_i, \bar{C}_j) = d(\bar{y}_i, \bar{y}_j)$ verwendet. Neben dem Clustering-Algorithmus hier vorgestellt, gibt es viele andere, für die wir den Leser auf [DHS01] verweisen. zum weiteren Studium.

8.8 Data Mining in der Praxis

Alle bisher vorgestellten Lernalgorithmen können als Werkzeuge für das Data Mining verwendet werden. Für den Benutzer ist es jedoch manchmal recht mühsam, sich an neue Software zu gewöhnen. Tools für jede Anwendung bereitzustellen und darüber hinaus die zu analysierenden Daten in die zu übertragen für jeden Einzelfall das passende Format.

Eine Reihe von Data-Mining-Systemen befasst sich mit diesen Problemen. Die meisten dieser Systeme bieten eine komfortable grafische Benutzeroberfläche mit vielfältigen Tools zur Visualisierung der Daten, zur Vorverarbeitung, z. B. Manipulation fehlender Werte, und zur Analyse. Für Zur Analyse kommen unter anderem die hier vorgestellten Lernalgorithmen zum Einsatz.

Besonders hervorzuheben ist die umfangreiche Open-Source-Java-Bibliothek WEKA. Es bietet eine große Anzahl an Algorithmen und vereinfacht die Entwicklung neuer Algorithmen.

Das frei verfügbare System KNIME, das wir im Folgenden kurz vorstellen

Der folgende Abschnitt bietet eine praktische Benutzeroberfläche und alle Arten von Tools oben erwähnt. KNIME verwendet auch WEKA-Module. Darüber hinaus bietet es eine

Einfache Möglichkeit, den Datenfluss der gewählten Visualisierungs-, Vorverarbeitungs- und Analysetools mit einem grafischen Editor zu steuern. Mittlerweile bieten eine Vielzahl anderer Systeme durchaus ähnliche Funktionalitäten, etwa das Open-Source-Projekt

RapidMiner (www.rapidminer.com), das System Clementine (www.spss.com/clementine), verkauft von SPSS, und das KXEN-Analyse-Framework (www.kxen.com).

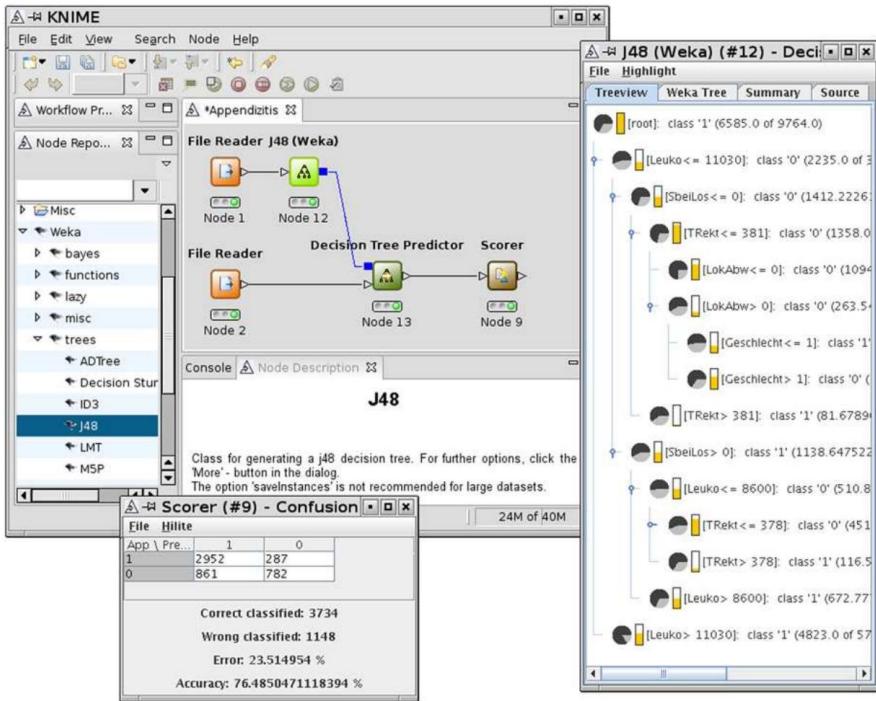


Abb. 8.35 Die KNIME-Benutzeroberfläche mit zwei zusätzlichen Ansichten, die den Entscheidungsbaum und die Verwirrungsmatrix anzeigen

8.8.1 Das Data-Mining-Tool KNIME

Anhand der LEXMED- Daten zeigen wir nun, wie man mit *KNIME* (Konstanz Information Miner, www.knime.org) Wissen aus Daten extrahieren kann. Zunächst erstellen wir einen Entscheidungsbaum wie in Abb. 8.35 dargestellt. Nach dem Anlegen eines neuen Projekts wird ein Workflow grafisch aufgebaut. Dazu werden die entsprechenden Tools einfach mit der Maus aus dem Node-Repository entnommen und in das Haupt-Workflow-Fenster gezogen.

Die Trainings- und Testdaten aus der C4.5-Datei können mit den beiden File-Reader-Knoten problemlos eingelesen werden. Diese Knoten können jedoch auch ganz einfach für andere Dateiformate konfiguriert werden. Die seitliche Ampel unter dem Knoten zeigt seinen Status an (nicht bereit, konfiguriert, ausgeführt). Anschließend wird Knoten J48 aus der WEKA-Bibliothek [WF01] ausgewählt, die eine Java-Implementierung von C4.5 enthält. Die Konfiguration hierfür ist recht einfach. Nun wird ein Prädiktorknoten ausgewählt, der den generierten Baum auf die Testdaten anwendet. Es fügt eine neue Spalte mit der vom Baum generierten Klassifizierung in die Testdatentabelle „Vorhersage“ ein. Daraus berechnet der Scorer-Knoten die in der Abbildung dargestellte Verwirrungsmatrix, die die Anzahl der korrekt klassifizierten Fälle für beide Klassen in der Diagonale und zusätzlich die Anzahl der falsch positiven und falsch negativen Datenpunkte angibt.

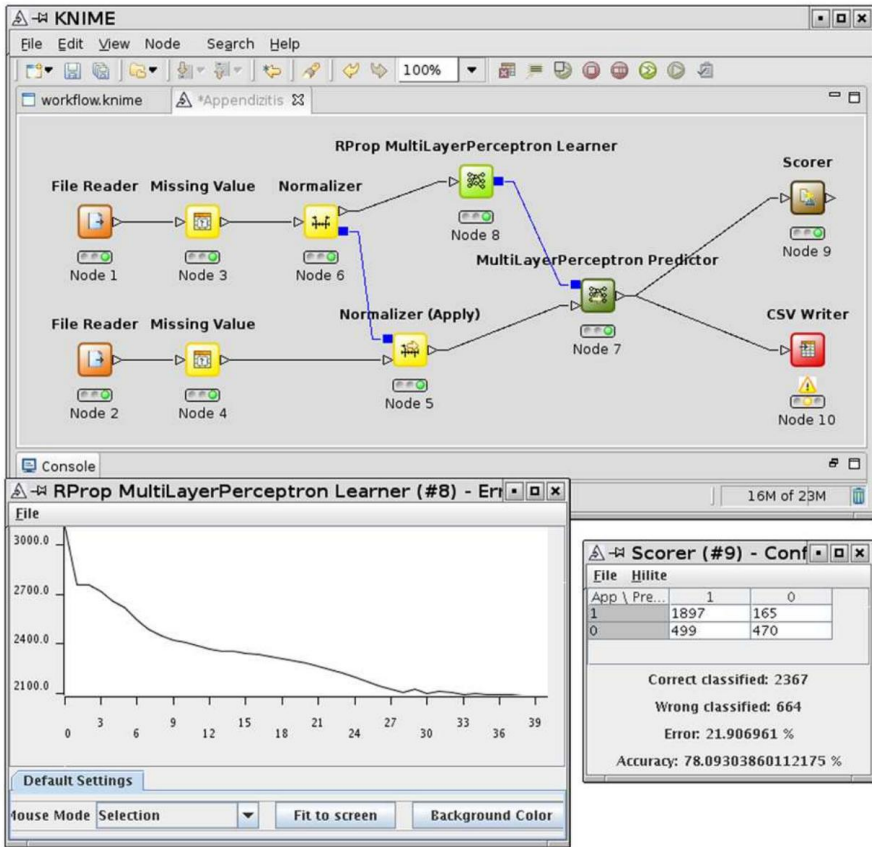


Abb. 8.36 Die KNIME-Benutzeroberfläche mit dem Workflow-Fenster, der Lernkurve und der Verwirrungsmatrix

Sobald der Fluss vollständig aufgebaut und alle Knoten konfiguriert sind, folgt ein beliebiger Knoten ausgeführt werden kann. Es stellt automatisch sicher, dass Vorgängerknoten ausgeführt werden, wenn notwendig. Der J48-Knoten generiert die rechts dargestellte Ansicht des Entscheidungsbaums der Figur. Dieser Baum ist identisch mit dem von C4.5 in Abschn. 8.4.5, obwohl hier der Knoten $T_{Rekt} \leq 378$ kollabiert dargestellt ist.

Zum Vergleich dient ein Projekt zum Erlernen eines mehrschichtigen Perzeptrons (siehe Abschn. 9.5). in Abb. 8.36 dargestellt. Dies funktioniert ähnlich wie das zuvor eingeführte lineare Perceptron, kann aber auch nicht linear trennbare Klassen unterteilen. Hier ist der Fluss etwas komplexer. Für die Fehlerbehandlung fehlender Werte ist ein zusätzlicher Knoten erforderlich Vorverarbeitung jeder der beiden Dateien. Wir haben es so eingestellt, dass diese Zeilen gelöscht werden. Da neuronale Netze nicht mit beliebigen Werten, den Werten aller Variablen, umgehen können werden mithilfe des „Normalizer“-Knotens linear in das Intervall $[0, 1]$ skaliert.

Nach der Anwendung des RProp-Lerners wurde eine Verbesserung der Backpropagation (siehe Sekt. 9.5) können wir den zeitlichen Verlauf des Approximationsfehlers analysieren die gezeigte Lernkurve. In der Verwirrungsmatrix gibt der Bewerter die Analyse aus der Testdaten. Der Knoten „CSV Writer“ unten rechts dient zum Exportieren der Ergebnisdateien, die dann extern verwendet werden können, um die in gezeigte ROC-Kurve zu generieren Abb. 7.10 auf Seite 142, für die es leider (noch) kein KNIME-Tool gibt.

Zusammenfassend können wir über KNIME (und ähnliche Tools) Folgendes sagen: z Projekte mit Datenanalyseanforderungen, die nicht zu exotisch sind, lohnt sich mit einer so leistungsstarken Workbench zur Datenanalyse zu arbeiten. Der Benutzer speichert bereits viel Zeit mit der Vorverarbeitungsphase. Es gibt eine große Auswahl an leicht verwendbaren „Mining-Tools“, wie z. B. Klassifikatoren für den nächsten Nachbarn, einfache Bayesian-Netzwerk-Lernalgorithmen sowie der k-Means-Clustering-Algorithmus (Abschn. 8.7.2). Auswertung der Ergebnisse, beispielsweise eine Kreuzvalidierung, können einfach durchgeführt werden. Es bleibt dabei Es sei erwähnt, dass es neben den gezeigten noch viele andere Tools zur Visualisierung der Daten gibt. Darüber hinaus haben die Entwickler von KNIME eine Erweiterung vorgenommen KNIME verfügbar, mit dem der Anwender eigene Tools in Java oder Python programmieren kann.

Es sollte jedoch auch erwähnt werden, dass die Benutzer dieser Art von Data Mining System sollte solide Vorkenntnisse über maschinelles Lernen und deren Nutzung mitbringen von Data-Mining-Techniken. Die Software allein kann keine Daten analysieren, wohl aber die Hand Für einen Spezialisten wird es zu einem leistungsstarken Werkzeug zur Gewinnung von Wissen aus Daten. Für den Einsteiger in das faszinierende Gebiet des maschinellen Lernens bietet sich ein solches System an eine ideale und einfache Möglichkeit, sein Wissen praktisch zu testen und zu vergleichen die verschiedenen Algorithmen. Der Leser kann dies in Übung 8.21 auf Seite 220 überprüfen.

8.9 Zusammenfassung

Wir haben mehrere Algorithmen aus dem etablierten Bereich des überwachten Lernens ausführlich behandelt, darunter Entscheidungsbaumlernen, Bayesianische Netzwerke und die nächstgelegenen Nachbarmethode. Diese Algorithmen sind in verschiedenen Anwendungen stabil und effizient einsetzbar und gehören somit zum Standardrepertoire in der KI und im Data Mining. Der Gleiches gilt für die Clustering-Algorithmen, die ohne „Supervisor“ und funktionieren finden sich beispielsweise in Suchmaschinenanwendungen. Verstärkungslernen da ein anderer Bereich des maschinellen Lernens ebenfalls keinen Vorgesetzten benötigt. Im Gegensatz zum überwachten Lernen, bei dem der Lernende die richtigen Aktionen oder Antworten als Etiketten erhält In den Trainingsdaten erhält man beim Reinforcement Learning nur hin und wieder positives oder negatives Feedback aus der Umgebung. Im Kap. 10 zeigen wir, wie das geht funktioniert. Nicht ganz so schwer ist die Aufgabe beim teilüberwachten Lernen, einem jungen Teilgebiet des maschinellen Lernens, wo nur sehr wenige aus einer großen Menge an Trainingsdaten vorhanden sind beschriftet.

Überwachtes Lernen ist mittlerweile ein etablierter Bereich mit vielen erfolgreichen Anwendungen. Für das überwachte Lernen von Daten mit kontinuierlichen Beschriftungen kann jeder Funktionsnäherungsalgorithmus verwendet werden. Daher gibt es viele Algorithmen aus verschiedenen Bereichen Mathematik und Informatik. In Abschn. 9 werden wir verschiedene vorstellen Arten neuronaler Netze, Algorithmen der kleinsten Quadrate und Support-Vektor-Maschinen,

die allesamt Funktionsnäherungen sind. Heutzutage erfreuen sich Gauß-Prozesse großer Beliebtheit, da sie sehr universell und einfach anzuwenden sind und dem Benutzer eine Schätzung der Unsicherheit der Ausgabewerte liefern [RW06].

Die folgende Taxonomie gibt einen Überblick über die wichtigsten Lernalgorithmen und deren Klassifizierung.

Überwachtes Lernen

- **Faules Lernen**
 - k-Nearest-Neighbor-Methode (Klassifizierung + Näherung)
 - Lokal gewichtete Regression (Näherung)
 - Fallbasiertes Lernen (Klassifizierung + Approximation) •
- Eager Learning** –
 - Entscheidungsbaum-Induktion (Klassifizierung)
 - Erlernen von Bayes'schen Netzwerken (Klassifizierung + Approximation)
 - Neuronale Netze (Klassifikation + Approximation)
 - Gaußsche Prozesse (Klassifikation + Näherung)
 - Wavelets, Splines, radiale Basisfunktionen, . . .

Unüberwachtes Lernen (Clustering)

- Nearest-Neighbor-Algorithmus
- Farthest-Neighbor-Algorithmus
- k-means •

Neuronale Netze

Reinforcement Learning

- Value Iteration •
- Q Learning •
- TD Learning •
- Policy-Gradient-Methoden •
- Neuronale Netze

Was über überwachtes Lernen gesagt wurde, ist jedoch nur wahr, wenn mit einem festen Satz bekannter Attribute gearbeitet wird. Ein interessantes, noch offenes Feld, auf dem intensiv geforscht wird, ist die automatische Funktionsauswahl. In Abschn. Für das Lernen mit Entscheidungsbäumen haben wir in Abschn. 8.4 einen Algorithmus zur Berechnung des Informationsgewinns von Attributen vorgestellt, der die Attribute nach ihrer Relevanz sortiert und nur diejenigen verwendet, die die Qualität der Klassifizierung verbessern. Mit dieser Art von Methode ist es möglich, die relevanten Attribute automatisch aus einer p Dieses Basisset muss jedoch manuell ausgewählt werden.

Noch offen und auch nicht klar definiert ist die Frage, wie die Maschine neue Attribute finden kann. Stellen wir uns einen Roboter vor, der Äpfel pflücken soll. Dazu muss er lernen, zwischen reifen und unreifen Äpfeln und anderen Gegenständen zu unterscheiden. Traditionell würden wir bestimmte Attribute wie die Farbe und Form von Pixelregionen bestimmen und dann einen Lernalgorithmus anhand manuell klassifizierter Bilder trainieren. Es ist auch möglich, dass beispielsweise ein neuronales Netz direkt mit allen Pixeln des Bildes als Eingabe trainiert wird, was bei hoher Auflösung allerdings mit erheblichen Rechenzeitproblemen verbunden ist. Wünschenswert wären hier Ansätze, die automatisch Vorschläge für relevante Features machen. Aber das ist immer noch Science-Fiction.

Clustering bietet einen Ansatz zur Merkmalsauswahl. Bevor wir die Apfelerkennungsmaschine trainieren, lassen wir das Clustering auf den Daten laufen. Für das (überwachte) Lernen der Klassen *Apple* und *Non Apple* sind die Eingabe nicht mehr alle Pixel, sondern nur noch die beim Clustering gefundenen Klassen, möglicherweise zusammen mit anderen Attributen. Clustering kann jedenfalls zur automatischen, kreativen „Entdeckung“ von Features genutzt werden. Es ist jedoch ungewiss, ob die entdeckten Merkmale relevant sind.

Noch schwieriger ist folgendes Problem: Gehen Sie davon aus, dass die zur Apfelerkennung verwendete Videokamera nur Schwarzweißbilder überträgt. Die Aufgabe lässt sich nicht mehr gut lösen. Es wäre schön, wenn die Maschine von sich aus kreativ wäre, indem sie beispielsweise vorschlägt, die Kamera durch eine Farbkamera zu ersetzen.

Das wäre heute zu viel verlangt.

Neben Fachwerken zu allen Teilgebieten des maschinellen Lernens gibt es die hervorragenden Lehrbücher [Mit97, Bis06, Alp04, DHS01, HTF09]. Aktuelle Forschungsergebnisse finden Sie im frei verfügbaren Journal of Machine Learning Research (<http://jmlr.csail.mit.edu>), im Machine Learning Journal sowie in den Tagungsbänden der International Conference on Machine Learning (ICML).) ist empfohlen. Für jeden Entwickler von Lernalgorithmen ist das Machine Learning Repository [DNM98] der University of California at Irvine (UCI) mit seiner großen Sammlung an Trainings- und Testdaten für Lernalgorithmen und Data-Mining-Tools interessant. MLOSS steht für Machine Learning Open Source Software und ist ein hervorragendes Verzeichnis mit Links zu frei verfügbarer Software (www.mloss.org).

8.10 Übungen

8.10.1 Einführung

Aufgabe 8.1

- (a) Geben Sie die Aufgabe eines Agenten an, der anhand der Messwerte für Temperatur, Luftdruck und Luftfeuchtigkeit das Wetter für den nächsten Tag vorhersagen soll. Das Wetter sollte in eine der drei Klassen eingeteilt werden: sonnig, bewölkt und regnerisch.
- (b) Beschreiben Sie den Aufbau einer Datei mit den Trainingsdaten für diesen Agenten.

Aufgabe 8.2 Zeigen Sie, dass die Korrelationsmatrix symmetrisch ist und dass alle Diagonalelemente gleich 1 sind.

8.10.2 Das Perzeptron

Übung 8.3 Wenden Sie die Perzeptron-Lernregel auf die Mengen an

$$M_+ = \{(0, 1, 8), (2, 0, 6)\} \text{ und } M_- = \{(1, 2, 1, 4), (0, 4, 1)\}$$

aus Beispiel 8.2 auf Seite 172 und geben Sie das Ergebnis der Werte des Gewichtsvektors an.

Übung 8.4 Gegeben sei die Tabelle rechts mit den Trainingsdaten: (a)

Zeigen Sie anhand eines Diagramms, dass die Daten

linear

trennbar sind. (b) Bestimmen Sie manuell die Gewichte

w_1 und w_2 sowie den Schwellenwert $\tilde{y}$ eines

Perzeptrons (mit Schwellenwert), das die Daten korrekt klassifiziert.

(c) Programmieren Sie die Perzeptron-Lernregel und wenden Sie Ihr Programm auf die Tabelle an.

Vergleichen Sie die ermittelten Gewichte mit den manuell berechneten.

Num. x_1 x_2 Klasse

1 610

2 730

3 820

4 900

5 841

6 861

7 921

8 951

Aufgabe 8.5

(a) Geben Sie eine visuelle Interpretation der heuristischen Initialisierung

$$w_0 = \frac{x_i \tilde{y}}{x_i \tilde{y} + x_i \tilde{y} M}$$

des in Abschn. 2.1 beschriebenen Gewichtsvektors [8.2.2](#).

(b) Geben Sie ein Beispiel für einen linear trennbaren Datensatz an, für den diese Heuristik keine Trennlinie erzeugt.

8.10.3 Methode des nächsten Nachbarn

Aufgabe 8.6

(a) Zeigen Sie das Voronoi-Diagramm für benachbarte Punktmengen.

(b) Zeichnen Sie dann die Klassentrennungslinien ein.



Aufgabe 8.7 Gegeben sei die Tabelle mit Trainingsdaten aus Aufgabe [8.4](#). Verwenden Sie im Folgenden den Manhattan-Abstand $d(a, b)$, definiert als $d(a, b) = |a_1 - b_1| + |a_2 - b_2|$, um den Abstand d zwischen zwei Datenpunkten $a = (a_1, a_2)$ und $b = (b_1, b_2)$ zu bestimmen. (a) Klassifizieren

Sie den Vektor $v = (8, 3.5)$ mit der Methode des nächsten Nachbarn. (b)

Klassifizieren Sie den Vektor $v = (8, 3.5)$ mit der k -Nächste-Nachbarn-Methode für $k = 2, 3, 5$.

Übung 8.8

(a) Zeigen Sie, dass es in einem zweidimensionalen Merkmalsraum sinnvoll ist, wie in [\(8.3\)](#) auf Seite [180](#) behauptet, die k nächsten Nachbarn mit dem Kehrwert des quadrierten Abstands zu gewichten.

(b) Warum sollte eine Gewichtung mit $w_i = \frac{1}{1 + \tilde{y} d(x, x_i)}$ weniger Sinn?

8.10.4 Entscheidungsbäume

Aufgabe 8.9 Zeigen Sie, dass die Definition $0 \log_2 0 := 0$ sinnvoll ist, mit anderen Worten, dass die Funktion $f(x) = x \log_2 x$ dadurch im Ursprung stetig wird.

Aufgabe 8.10 Bestimmen Sie die Entropie für die folgenden Verteilungen.

- (a) $(1, 0, 0, 0, 0)$
- (B) $\frac{1}{2}, \frac{1}{2}, 0, 0, 0$
- (C) $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0, 0$
- (D) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}$
- (e) $\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}$
- (F) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \dots$

Aufgabe 8.11

(a) Zeigen Sie, dass sich die beiden unterschiedlichen Definitionen der Entropie aus (7.9) auf Seite 124 und Definition 8.4 auf Seite 188 nur um einen konstanten Faktor unterscheiden, nämlich das

$$\sum_{i=1}^N p_i \log_2 p_i = c$$

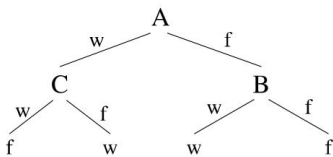
$$\sum_{i=1}^N p_i \ln p_i$$

und geben Sie die Konstante
c an. (b) Zeigen Sie, dass es für die MaxEnt-Methode und für Entscheidungsbäume keinen Unterschied macht, welche der beiden Formeln wir verwenden.

Aufgabe 8.12 Entwickeln Sie einen Entscheidungsbaum für den Datensatz D aus Übung 8.4 auf Seite 217.

(a) Behandeln Sie beide Attribute als diskret. (b) Behandeln Sie nun das Attribut x2 als kontinuierlich und x1 als diskret. (c) Lassen Sie C4.5 einen Baum mit beiden Varianten generieren. Verwenden Sie -m 1 -t 10 als Parameter, um unterschiedliche Vorschläge zu erhalten.

Übung 8.13 Gegeben seien der folgende Entscheidungsbaum und die folgenden Tabellen für Trainings- und Testdaten:



Trainingsdaten	Testdaten
ABC-Klasse	ABC-Klasse
ttft	ttft
ttft	ttft
tttf	ttft
ffft	ftff
ffff	ffft
fttt	

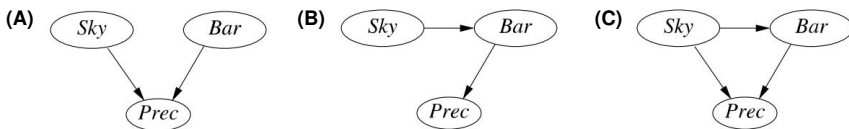
(a) Geben Sie die Korrektheit des Baums für die Trainings- und Testdaten an. (b) Geben Sie eine Aussagenlogikformel an, die dem Baum entspricht. (c) Beschneiden Sie den Baum, zeichnen Sie den resultierenden Baum und geben Sie seine Richtigkeit für die Trainings- und Testdaten.

Übung 8.14

- (a) Der Algorithmus zur Generierung von Entscheidungsbäumen (Abb. 8.26 auf Seite 193) eliminiert bei der Ermittlung des aktuellen Attributs nicht die Attribute, die bereits weiter oben im Baum verwendet wurden. Trotzdem kommt ein diskretes Attribut in einem Pfad höchstens einmal vor. Warum?
- (b) Warum können kontinuierliche Attribute mehrfach vorkommen?

8.10.5 Lernen von Bayes'schen Netzwerken

Aufgabe 8.15 Verwenden Sie die in Übung 7.3 auf Seite 158 angegebene Verteilung und bestimmen Sie die CPTs für die drei Bayes'schen Netzwerke:



- (d) Bestimmen Sie die Verteilung für die beiden Netzwerke aus (a) und (b) und vergleichen Sie diese mit der Originalverteilung. Welches Netzwerk ist „besser“?
- (e) Bestimmen Sie nun die Verteilung für Netzwerk (c). Was fällt Ihnen ein? Justify!

Übung 8.16 Zeigen Sie, dass für binäre Variablen $S_1, \dots, S_n$ und die binäre Klassenvariable K ein linearer Score der Form vorliegt

$$\text{Entscheidung} = \begin{cases} \text{Positiv, wenn } w_1 S_1 + \dots + w_n S_n > \tilde{y}, & \text{sonst} \\ \text{negativ} \end{cases}$$

ist in Bezug auf das Perzeptron und den naiven Bayes-Klassifikator gleichermaßen ausdrucksstark, die beide nach der Formel entscheiden

$$\text{Entscheidung} = \begin{cases} \text{positiv, wenn } P(K|S_1, \dots, S_n) > 1/2, & \text{sonst} \\ \text{negativ} \end{cases}$$

Übung 8.17 Bei der Implementierung der Textklassifizierung mit Naive Bayes kann es schnell zu einem Exponentenunterlauf kommen, da die Faktoren $P(w_i|K)$ (die in (8.10) auf Seite 206 erscheinen) typischerweise alle sehr klein sind, was dazu führen kann, dass sie extrem klein sind Ergebnisse. Wie können wir dieses Problem entschärfen?

Übung 8.18 Schreiben Sie ein Programm zur naiven Bayes-Textanalyse. Anschließend trainieren und testen Sie es anhand von Text-Benchmarks mit einem Tool Ihrer Wahl. Das Zählen der Häufigkeit von Wörtern im Text kann unter Linux mit dem Befehl ganz einfach durchgeführt werden

```
cat <datei> | tr -d „[:punct:]“ | tr -s „[:space:]“ „\n“ | sortieren | uniq -ci
```

Erhalten Sie die Twenty Newsgroups-Daten von Tom Mitchell in der Benchmark-Sammlung für maschinelles Lernen der UCI (Machine Learning Repository) [DNM98]. Dort finden Sie auch einen Hinweis auf ein naives Bayes-Programm zur Textklassifizierung von Mitchell.

8.10.6 Clustering

Aufgabe 8.19 Zeigen Sie, dass für Algorithmen, die nur Distanzen vergleichen, die Anwendung einer streng monoton steigenden Funktion f auf die Distanz keinen Unterschied macht. Mit anderen Worten müssen Sie zeigen, dass der Abstand $d_1(x, y)$ und der Abstand $d_2(x, y) := f(d_1(x, y))$ bezüglich der Ordnungsrelation zum gleichen Ergebnis führen.

Aufgabe 8.20 Bestimmen Sie die Abstände ds (Skalarprodukt) der folgenden Texte zueinander. x_1 : Wir werden kurz

die Anwendung von Naive Bayes auf die Textanalyse vorstellen

Beispieltext von Alan Turing aus [Tur50].

x_2 : Wir können hoffen, dass Maschinen irgendwann überhaupt mit Männern konkurrieren werden geistige Felder. Aber welche sind die besten für den Anfang?

x_3 : Auch hier weiß ich nicht, was die richtige Antwort ist, aber ich denke, beide Ansätze sollten es tun müde sein.

8.10.7 Data Mining

Übung 8.21 Verwenden Sie KNIME (www.knime.de) und (a)

Laden Sie die Beispieldatei mit den Iris-Daten aus dem KNIME-Verzeichnis und experimentieren Sie mit den verschiedenen Datendarstellungen, insbesondere mit den Streudiagrammen.

(b) Trainieren Sie zunächst einen Entscheidungsbaum für die drei Klassen und dann ein RProp-

Netzwerk. (c) Laden Sie die Blinddarmentzündungsdaten auf die Website dieses Buches. Vergleichen Sie die Klassifizierungsqualität der Methode „k nächster Nachbar“ mit der eines RProp-Netzwerks. Optimieren Sie k sowie die Anzahl der versteckten Neuronen des RProp-

Netzwerks. (d) Besorgen Sie sich einen Datensatz Ihrer Wahl aus der UCI-Datensammlung für Data Mining unter <http://kdd.ics.uci.edu> oder für maschinelles Lernen unter <http://mllearn.ics.uci.edu/MLRepository.html> und experimentiere damit.

Neuronale Netze sind Netzwerke von Nervenzellen im Gehirn von Menschen und Tieren. Das menschliche Gehirn verfügt über etwa 100 Milliarden Nervenzellen. Wir Menschen verdanken unsere Intelligenz und unsere Fähigkeit, verschiedene motorische und intellektuelle Fähigkeiten zu erlernen, den komplexen Relais und der Adaptivität des Gehirns. Seit vielen Jahrhunderten versuchen Biologen, Psychologen und Ärzte zu verstehen, wie das Gehirn funktioniert. Um 1900 kam die revolutionäre Erkenntnis, dass diese winzigen physischen Bausteine des Gehirns, die Nervenzellen und ihre Verbindungen, für Bewusstsein, Assoziationen, Gedanken, Bewusstsein und Lernfähigkeit verantwortlich sind.

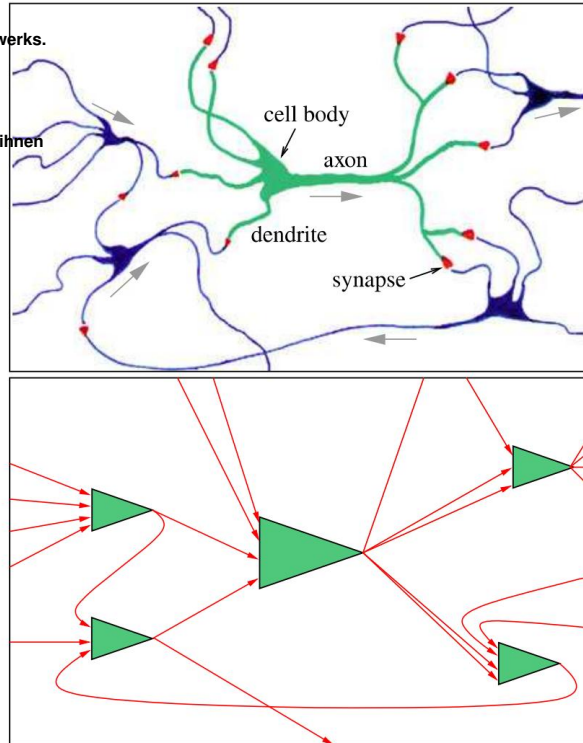
Der erste große Schritt hin zu neuronalen Netzen in der KI wurde 1943 von McCulloch und Pitts in einem Artikel mit dem Titel „Ein logischer Kalkül der der Nervenaktivität immanenten Ideen“ gemacht [AR88]. Sie waren die ersten, die ein mathematisches Modell des Neurons als grundlegendes Schaltelement des Gehirns vorstellten. Dieser Artikel legte den Grundstein für den Aufbau künstlicher neuronaler Netze und damit für diesen sehr wichtigen Zweig der KI.

Wir könnten den Bereich der Modellierung und Simulation neuronaler Netze als den Zweig der Bionik innerhalb der KI betrachten.¹ Nahezu alle Bereiche der KI versuchen, kognitive Prozesse nachzubilden, beispielsweise in der Logik oder im probabilistischen Denken. Allerdings haben die zur Modellierung verwendeten Werkzeuge – nämlich Mathematik, Programmiersprachen und digitale Computer – nur sehr wenig mit dem menschlichen Gehirn gemein. Bei künstlichen neuronalen Netzen ist der Ansatz anders. Ausgehend von Erkenntnissen über die Funktion natürlicher neuronaler Netze versuchen wir, diese in Hardware zu modellieren, zu simulieren und sogar zu rekonstruieren. Jeder Forscher auf diesem Gebiet steht vor der faszinierenden und spannenden Herausforderung, Ergebnisse mit

In diesem Kapitel werden wir versuchen, den historischen Fortschritt zu skizzieren, indem wir ausgehend von den wichtigsten biologischen Erkenntnissen ein Modell des Neurons und seiner Interkonnektivität definieren. Anschließend stellen wir einige wichtige und grundlegende Modelle vor: das Hopfield-Modell, zwei einfache assoziative Speichermodelle und den – in der Praxis überaus wichtigen – Backpropagation-Algorithmus.

¹Bionics beschäftigt sich mit der Erschließung der „Entdeckungen der belebten Natur“ und ihrer innovativen Umsetzung in Technologie [Wik10].

Abb. 9.1 Zwei Phasen der Modellierung eines neuronalen Netzwerks. Oben ein biologisches Modell und unten ein formales Modell mit Neuronen und gerichteten Verbindungen zwischen ihnen



9.1 Von der Biologie zur Simulation

Jedes der rund 100 Milliarden Neuronen im menschlichen Gehirn hat, wie in Abb. 9.1 vereinfacht dargestellt, die folgende Struktur und Funktion. Neben dem Zellkörper verfügt das Neuron über ein Axon, das über die Dendriten lokale Verbindungen zu anderen Neuronen herstellen kann. Das Axon kann jedoch bis zu einem Meter lang in Form einer Nervenfaser durch den Körper wachsen.

Der Zellkörper des Neurons kann kleine elektrische Ladungen speichern, ähnlich wie ein Kondensator oder eine Batterie. Dieser Speicher wird durch eingehende elektrische Impulse anderer Neuronen geladen. Je mehr elektrische Impulse eingehen, desto höher ist die Spannung. Wenn die Spannung einen bestimmten Schwellenwert überschreitet, feuert das Neuron. Das bedeutet, dass es seinen Vorrat entlädt, indem es einen Impuls über das Axon und die Synapsen sendet. Der elektrische Strom teilt sich und gelangt über die Synapsen zu vielen anderen Neuronen, in denen der gleiche Vorgang abläuft.

Nun stellt sich die Frage nach der Struktur des neuronalen Netzwerks. Jedes der etwa 1011 Neuronen im Gehirn ist mit etwa 1000 bis 10.000 anderen Neuronen verbunden, was insgesamt über 10^{14} Verbindungen ergibt. Wenn wir weiter bedenken, dass diese gigantische Anzahl extrem dünner Verbindungen aus weichem, dreidimensionalem Gewebe besteht und Experimente am menschlichen Gehirn nicht einfach durchzuführen sind, dann wird klar, warum wir keinen detaillierten Schaltplan davon haben das Gehirn. Vermutlich

Wir werden niemals in der Lage sein, den Schaltplan unseres Gehirns allein aufgrund seiner immensen Größe vollständig zu verstehen.

Aus heutiger Sicht lohnt es sich nicht mehr, einen vollständigen Schaltplan des Gehirns zu erstellen, da die Struktur des Gehirns adaptiv ist. Es verändert sich spontan und passt sich den Aktivitäten und Umwelteinflüssen des Einzelnen an. Die zentrale Rolle spielen dabei die Synapsen, die die Verbindung zwischen Neuronen herstellen. An der Verbindungsstelle zwischen zwei Neuronen ist es, als ob sich zwei Kabel treffen. Allerdings sind die beiden Leitungen nicht perfekt leitend miteinander verbunden, vielmehr besteht eine kleine Lücke, über die die Elektronen nicht direkt springen können. Diese Lücke wird mit chemischen Substanzen, sogenannten Neurotransmittern, gefüllt. Diese können durch eine angelegte Spannung ionisiert werden und dann eine Ladung über die Lücke transportieren. Die Leitfähigkeit dieser Lücke hängt von vielen Parametern ab, beispielsweise der Konzentration und der chemischen Zusammensetzung des Neurotransmitters. Aufschlussreich ist, dass die Funktion des Gehirns sehr empfindlich auf Veränderungen dieser synaptischen Verbindung reagiert, beispielsweise durch den Einfluss von Alko

Wie funktioniert das Lernen in einem solchen neuronalen Netzwerk? Das Überraschende dabei ist, dass nicht die eigentlichen aktiven Einheiten, nämlich die Neuronen, adaptiv sind, sondern die Verbindungen zwischen ihnen, also die Synapsen. Konkret kann sich dadurch ihre Leitfähigkeit verändern. Wir wissen, dass eine Synapse stärker wird, je mehr elektrischer Strom sie transportieren muss. Stärker bedeutet hier, dass die Synapse eine höhere Leitfähigkeit aufweist. Die genutzten Synapsen erhalten oft ein immer höheres Gewicht. Bei selten genutzten oder gar nicht aktiven Synapsen nimmt die Leitfähigkeit weiter ab. Dies kann sogar zu ihrem Absterben führen.

Alle Neuronen im Gehirn arbeiten asynchron und parallel, allerdings im Vergleich zu einem Computer mit sehr geringer Geschwindigkeit. Die Zeit für einen Nervenimpuls beträgt etwa eine Millisekunde, genau die Zeit, die die Ionen für den Transport über den synaptischen Spalt benötigen. Die Taktfrequenz des Neurons liegt dann unter einem Kilohertz und ist damit um den Faktor 10⁶ niedriger als bei modernen Computern. Dieser Nachteil wird jedoch bei vielen komplexen kognitiven Aufgaben, etwa der Bilderkennung, durch die sehr hohe Taktfrequenz mehr als ausgeglichen. Grad der Parallelverarbeitung im Netzwerk von Nervenzellen.

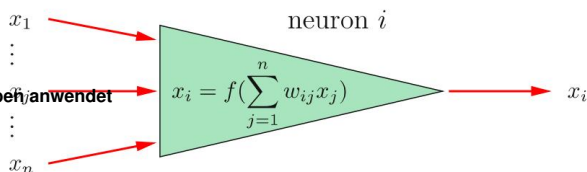
Die Verbindung zur Außenwelt erfolgt über sensorische Neuronen, zum Beispiel auf der Netzhaut in den Augen, oder über Nervenzellen mit sehr langen Axonen, die vom Gehirn bis zu den Muskeln reichen und so Aktionen wie die Bewegung eines Menschen ausführen können. Bein.

Allerdings ist noch unklar, wie die diskutierten Prinzipien intelligentes Verhalten ermöglichen. Wie viele Forscher in den Neurowissenschaften werden wir versuchen, anhand von Simulationen eines einfachen mathematischen Modells zu erklären, wie kognitive Aufgaben, beispielsweise die Mustererkennung, möglich werden.

9.1.1 Das mathematische Modell

Zuerst ersetzen wir die kontinuierliche Zeitachse durch eine diskrete Zeitskala. Das Neuron i führt die folgende Berechnung in einem Zeitschritt durch. Das „Laden“ der Aktivierung

Abb. 9.2 Die Struktur eines formalen Neurons, das die Aktivierungsfunktion f auf die gewichtete Summe aller Eingaben anwendet



Die Berechnung des Potentials erfolgt einfach durch Summation der gewichteten Ausgangswerte $x_1, \dots, x_n$ aller eingehenden Verbindungen über die Formel

$$\sum_{j=1}^N w_{ij} x_j.$$

Diese gewichtete Summe wird von den meisten neuronalen Modellen berechnet. Darauf wird dann eine **Aktivierungsfunktion** f angewendet und das Ergebnis ermittelt

$$x_i = f\left(\sum_{j=1}^N w_{ij} x_j\right)$$

wird als Ausgabe über die synaptischen Gewichte an die benachbarten Neuronen weitergegeben. In Abb. 9.2 ist ein solches modelliertes Neuron dargestellt. Für die Aktivierungsfunktion gibt es mehrere Möglichkeiten. Am einfachsten ist die Identität: $f(x) = x$. Das Neuron berechnet also nur die gewichtete Summe der Eingabewerte und gibt diese weiter. Dies führt jedoch häufig zu Konvergenzproblemen mit der neuronalen Dynamik, da die Funktion $f(x) = x$ unbegrenzt ist und die Funktionswerte mit der Zeit über alle Grenzen hinaus wachsen können.

Sehr gut eingeschränkt ist dagegen die Schwellenfunktion (Heaviside-Stufenfunktion)

$$H_{\tilde{y}}(x) = \begin{cases} 0 & \text{wenn } x < \tilde{y}, \\ 1 & \text{sonst.} \end{cases}$$

Das gesamte Neuron berechnet dann seine Ausgabe als

$$x_i = \begin{cases} 0 & \text{wenn } \sum_{j=1}^N w_{ij} x_j < \tilde{y}, \\ 1 & \text{sonst.} \end{cases}$$

Diese Formel ist identisch mit (8.1) auf Seite 173, also mit einem Perzeptron mit der Schwelle $\tilde{y}$ (Abb. 9.3 auf Seite 225). Die Eingabeneuronen $1, \dots, n$ haben hier nur die Funktion von Variablen, die ihre von außen vorgegebenen Werte $x_1, \dots, x_n$ unverändert weitergeben.

Für binäre Neuronen ist die Schrittfunktion durchaus sinnvoll, da die Aktivierung eines Neurons ohnehin nur die Werte Null oder Eins annehmen kann. Im Gegensatz dazu für kontinuierlich

Abb. 9.3 Das Neuron mit einer Stufenfunktion funktioniert wie ein Perzeptron mit einer Schwelle

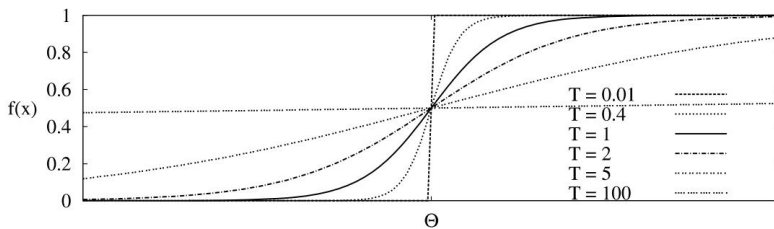
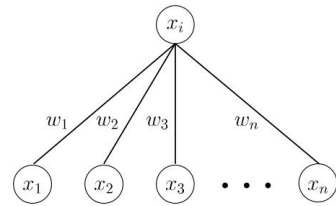


Abb. 9.4 Die Sigmoidfunktion für verschiedene Werte des Parameters T . Das sehen wir im Limit 0 ergibt die Sprungfunktion

Bei Neuronen mit Aktivierungen zwischen 0 und 1 erzeugt die Schrittfunktion eine Diskontinuität. Dies kann jedoch durch eine **Sigmoidfunktion** geglättet werden, z

$$f(x) = \frac{1}{1 + e^{-\frac{x - \bar{y}}{T}}}$$

mit der Grafik in Abb. 9.4. In der Nähe des kritischen Bereichs um den Schwellenwert $\bar{y}$ verhält sich diese Funktion nahezu linear und hat einen asymptotischen Grenzwert. Die Glättung kann durch den Parameter T variiert werden

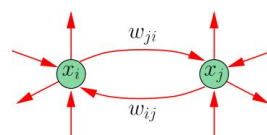
Die Modellierung des Lernens ist von zentraler Bedeutung für die Theorie neuronaler Netze. Wie bereits erwähnt, besteht eine Möglichkeit des Lernens darin, eine Synapse entsprechend zu stärken, je nachdem wie viele elektrische Impulse sie übertragen muss. Dieses Prinzip wurde 1949 von D. Hebb postuliert und ist als **Hebb-Regel bekannt**:

Wenn eine Verbindung w_{ij} zwischen Neuron j und Neuron i besteht und wiederholte Signale von Neuron j an Neuron i gesendet werden, was dazu führt, dass beide Neuronen gleichzeitig aktiv sind, dann wird das Gewicht w_{ij} verstärkt. Eine mögliche Formel für die Gewichtsänderung w_{ij} lautet

$$\Delta w_{ij} = \eta x_i x_j$$

mit der Konstanten η (Lernrate), die die Größe der einzelnen Lernschritte bestimmt.

Abb. 9.5 Wiederkehrende
Verbindungen zwischen zwei
Neuronen in einem
Hopfield-Netzwerk



Es gibt viele Modifikationen dieser Regel, die dann zu unterschiedlichen Arten von Netzwerken oder Lernalgorithmen führen. In den folgenden Abschnitten werden wir einige davon kennenlernen.

9.2 Hopfield-Netzwerke

Wenn wir uns die Hebb-Regel ansehen, sehen wir, dass bei Neuronen mit Werten zwischen Null und Eins die Gewichte nur mit der Zeit zunehmen können. Nach dieser Regel ist es nicht möglich, dass ein Neuron schwächer wird oder gar stirbt. Dies kann beispielsweise durch eine Zerfallskonstante modelliert werden, die ein ungenutztes Gewicht pro Zeitschritt um einen konstanten Faktor abschwächt, beispielsweise 0,99.

Ganz anders löst dieses Problem das von Hopfield 1982 vorgestellte Modell [Hop82]. Es verwendet binäre Neuronen, jedoch mit den beiden Werten $\tilde{y}1$ für inaktiv und 1 für aktiv. Mit der Hebb-Regel erhalten wir immer dann einen positiven Beitrag zum Gewicht, wenn zwei Neuronen gleichzeitig aktiv sind. Ist jedoch nur eines der beiden Neuronen aktiv, ist w_{ij} negativ.

Auf dieser Idee basieren *Hopfield-Netzwerke*, die ein schönes und anschauliches Beispiel für *autoassoziatives Gedächtnis* darstellen. Muster können im autoassoziativen Speicher gespeichert werden. Um ein gespeichertes Muster aufzurufen, reicht es aus, ein ähnliches Muster bereitzustellen. Der Shop findet dann das ähnlichste gespeicherte Muster. Eine klassische Anwendung hierfür ist die Handschrifterkennung.

In der Lernphase eines Hopfield-Netzwerks werden N binär codierte Muster gespeichert. Die Vektoren $q^1, \dots, q^N$, j sollen gelernt werden. Jede Komponente $q^j_i \in \{\tilde{y}1, 1\}$ eines solchen Vektors q repräsentiert ein Pixel eines Musters. Für Vektoren, die aus n Pixeln bestehen, wird ein neuronales Netzwerk mit n Neuronen verwendet, eines für jede Pixelposition. Die Neuronen sind vollständig verbunden mit der Einschränkung, dass die Gewichtsmatrix symmetrisch ist und alle diagonalen Elemente null sind. Das heißt, es gibt keine Verbindung zwischen einem Neuron und sich selbst.

Das vollständig vernetzte Netzwerk beinhaltet komplexe Rückkopplungsschleifen, sogenannte *Rekursionen*, im Netzwerk (Abb. 9.5).

N Muster können gelernt werden, indem einfach alle Gewichte mit der Formel berechnet werden

$$w_{ij} = \frac{1}{N} \sum_{k=1}^N q^k_i q^k_j. \quad (9.1)$$

Diese Formel weist auf eine interessante Beziehung zur Hebb-Regel hin. Jedes Muster, in dem die Pixel i und j den gleichen Wert haben, trägt positiv zum Gewicht w_{ij} bei.

Jedes andere Muster leistet einen negativen Beitrag. Da jedes Pixel

entspricht einem Neuron, hier werden die Gewichte zwischen Neuronen, die gleichzeitig den gleichen Wert haben, verstärkt. Bitte beachten Sie diesen kleinen Unterschied zur Hebb-Regel.

Sind alle Muster gespeichert, kann das Netzwerk zur Mustererkennung genutzt werden. Wir geben dem Netzwerk ein neues Muster x und aktualisieren die Aktivierungen aller Neuronen in einem asynchronen Prozess gemäß der Regel

$$x_i = \begin{cases} 1 & \text{wenn} \sum_{j=1}^N w_{ij} x_j > 0, \\ 0 & \text{anders} \end{cases} \quad (9.2)$$

bis das Netzwerk stabil wird, also bis sich keine Aktivierungen mehr ändern. Als Programmschema lautet dies wie folgt:

HOPFIELDASSOCIATOR(q)

Alle Neuronen initialisieren: $x =$

q

Wiederholen $i = \text{Random}(1, n)$

Aktualisieren Sie Neuron i gemäß (9.2)

Bis x konvergiert

Rückgabe (x)

9.2.1 Anwendung auf ein Mustererkennungsbeispiel

Wir wenden den beschriebenen Algorithmus auf ein einfaches Mustererkennungsbeispiel an. Es sollte Ziffern in einem 10×10 Pixel großen Feld erkennen. Das Hopfield-Netzwerk verfügt somit über 100 Neuronen mit insgesamt

$$\frac{100 \cdot 99}{2} = 4950$$

Gewichte. Zunächst werden die Muster der Ziffern 1, 2, 3, 4 in Abb. 9.6 auf Seite 228 oben trainiert. Das heißt, die Gewichte werden nach (9.1) auf Seite 226 berechnet. Dann fügen wir das Muster mit zusätzlichem Rauschen ein und lassen die Hopfield-Dynamik bis zur Konvergenz laufen. In den Zeilen 2 bis 4 der Abbildung sind fünf Momentaufnahmen der Netzwerkentwicklung während der Erkennung dargestellt. Bei 10 % Rauschen werden alle vier gelernten Muster sehr zuverlässig erkannt. Oberhalb von etwa 20 % Rauschen konvergiert der Algorithmus häufig mit anderen gelernten Mustern oder sogar mit Mustern, die nicht gelernt wurden. Mehrere solcher Muster sind in Abb. 9.6 auf Seite 228 unten dargestellt.

Nun speichern wir die Ziffern 0 bis 9 (Abb. 9.7 auf Seite 229 oben) im selben Netzwerk und testen das Netzwerk erneut mit Mustern, die einen zufälligen Anteil von etwa 10 % invertierten Pixeln haben. In der Abbildung sehen wir deutlich, dass die Hopfield-Iteration selbst bei nur 10 % Rauschen oft nicht zum ähnlichsten erlernten Zustand konvergiert. Offensichtlich

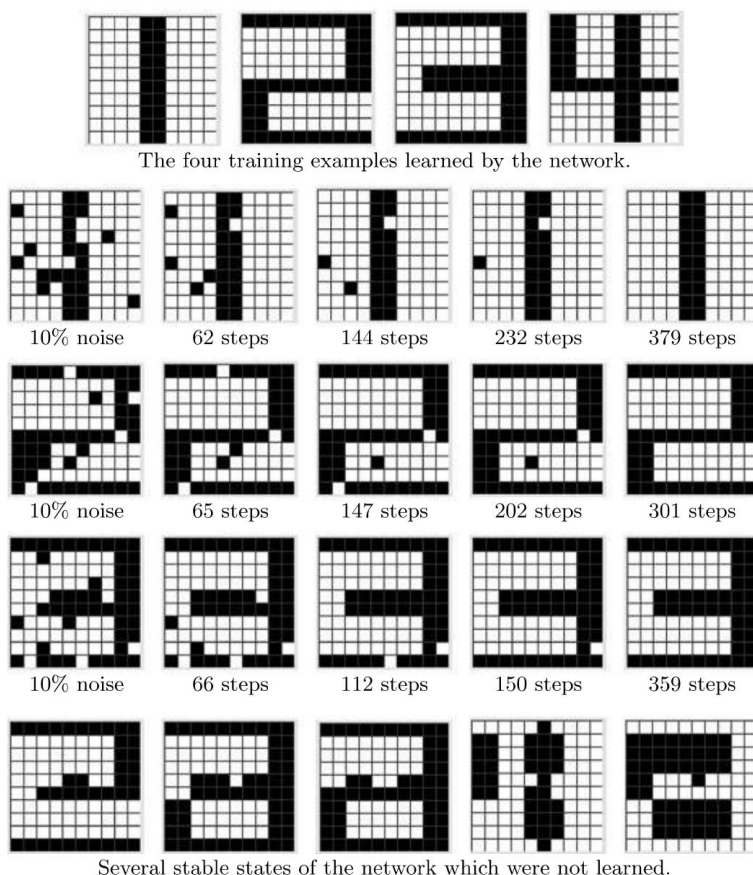


Abb. 9.6 Dynamik eines Hopfield-Netzwerks. In den Zeilen 2, 3 und 4 können wir gut erkennen, wie das Netzwerk konvergiert und das gelernte Muster nach etwa 300 bis 400 Iterationen erkannt wird. In der letzten Zeile werden mehrere stabile Zustände angezeigt, die das Netzwerk erreicht, wenn das Eingabemuster zu stark von allen gelernten Mustern abweicht

Das Netzwerk kann vier Muster sicher speichern und erkennen, für zehn Muster ist jedoch die Speicherkapazität überschritten. Um dies besser zu verstehen, werfen wir einen kurzen Blick auf die Theorie dieses Netzwerks.

9.2.2 Analyse

1982 zeigte John Hopfield in [Hop82], dass dieses Modell formal einem physikalischen Modell des Magnetismus entspricht. Kleine Elementarmagnete, sogenannte Spins, beeinflussen sich gegenseitig über ihre Magnetfelder (siehe Abb. 9.8 auf Seite 229). Wenn wir

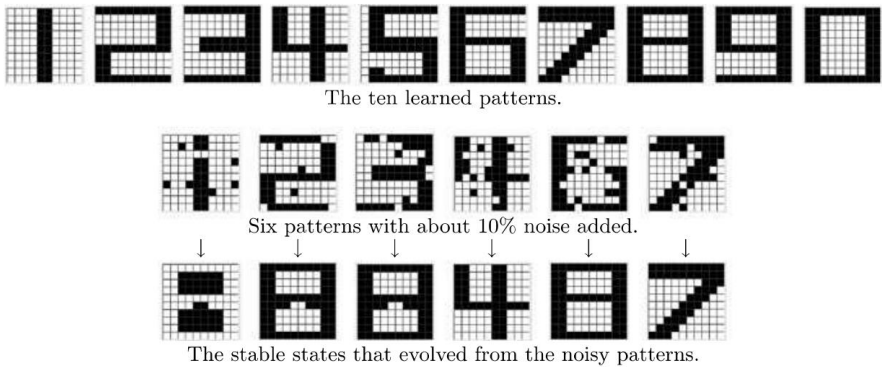


Abb. 9.7 Für zehn erlernte Zustände zeigt das Netzwerk chaotisches Verhalten. Selbst bei geringem Rauschen konvergiert das Netzwerk mit falschen Mustern oder Artefakten

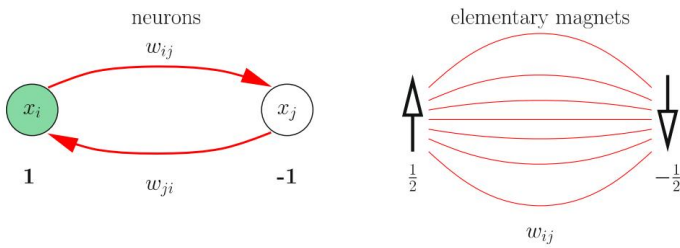


Abb. 9.8 Vergleich zwischen der neuronalen und physikalischen Interpretation des Hopfield-Modells

Beobachten Sie zwei solcher Spins i und j , sie interagieren über eine Konstante w_{ij} und die Gesamtenergie des Systems beträgt dann

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j .$$

Übrigens gilt $w_{ii} = 0$ auch in der Physik, weil Teilchen keine Selbstwechselwirkung haben. Da physikalische Wechselwirkungen symmetrisch sind, gilt $w_{ij} = w_{ji}$.

Ein physikalisches System im Gleichgewicht nimmt einen (stabilen) Zustand minimaler Energie ein und minimiert somit $E(x, y)$. Wird ein solches System in einen beliebigen Zustand gebracht, dann bewegt es sich in einen Zustand minimaler Energie. Die in (9.2) auf Seite 227 definierte Hopfield-Dynamik entspricht genau diesem Prinzip, da sie den Zustand in jeder Iteration so aktualisiert, dass von den beiden Zuständen $\tilde{y}_i$ und 1 derjenige mit der kleineren Gesamtenergie eingenommen wird. Der Beitrag des Neurons i zur Gesamtenergie beträgt

$$-\frac{1}{2} \sum_{j=1}^N w_{ij} x_j .$$

Wenn jetzt

$$\sum_{j=1}^N w_{ij} x_j < 0,$$

dann führt $x_i = -1$ zu einem negativen Beitrag zur Gesamtenergie und $x_i = 1$ zu einem positiven Beitrag. Für $x_i = -1$ nimmt das Netzwerk einen Zustand niedrigerer Energie ein als für $x_i = 1$. Analog können wir das im Fall von behaupten

$$\sum_{j=1}^N w_{ij} x_j \geq 0,$$

es muss wahr sein, dass $x_i = 1$.

Wenn jede einzelne Iteration der neuronalen Dynamik zu einer Verringerung der Energiefunktion führt, dann nimmt die Gesamtenergie des Systems monoton mit der Zeit ab. Da es nur endlich viele Zustände gibt, bewegt sich das Netzwerk mit der Zeit in einen Zustand minimaler Energie. Nun stellt sich uns die spannende Frage: Was bedeuten diese Minima der Energiefunktion?

Wie wir im Mustererkennungsexperiment gesehen haben, konvergiert das System bei wenigen gelernten Mustern zu einem der gelernten Muster. Die erlernten Muster repräsentieren Minima der Energiefunktion im Zustandsraum. Werden jedoch zu viele Muster gelernt, konvergiert das System zu Minima, die nicht den gelernten Mustern entsprechen. Hier haben wir einen Übergang von einer geordneten Dynamik in eine chaotische eins.

Hopfield und andere Physiker haben genau diesen Prozess untersucht und gezeigt, dass es bei einer kritischen Anzahl erlernter Muster tatsächlich zu einem Phasenübergang kommt. Übersteigt die Anzahl der gelernten Muster diesen Wert, dann geht das System von der geordneten Phase in die chaotische Phase über.

In der magnetischen Physik gibt es einen solchen Übergang vom ferromagnetischen Modus, bei dem alle Elementarmagnete versuchen, sich parallel auszurichten, zu einem sogenannten Spinglas, bei dem die Spins chaotisch interagieren. Ein besser sichtbares Beispiel für einen solchen physikalischen Phasenübergang ist das Schmelzen eines Eiskristalls. Der Kristall befindet sich in einem hohen Ordnungszustand, da die H₂O-Moleküle streng geordnet sind. Im flüssigen Wasser hingegen ist die Struktur der Moleküle aufgelöst und ihre Position

In einem neuronalen Netz kommt es dann zu einem Phasenübergang vom geordneten Lernen und Erkennen von Mustern zum chaotischen Lernen bei zu vielen Mustern, die nicht mehr sicher erkannt werden können. Hier sehen wir durchaus Parallelen zu Wirkungen, die wir gelegentlich selbst erlebt haben.

Wir können diesen Phasenübergang [RMS92] verstehen, wenn wir alle Neuronen in a Musterzustand, zum Beispiel ± 1 , bringen und die gelernten Gewichte aus (9.1) auf Seite 226 in den Begriff $\sum_{j=1}^N w_{ij} x_j$ einfügen, was für die Aktualisierung von Neuron i relevant ist. Daraus ergibt sich $\sum_{j=1}^N w_{ij} x_j$.

$$\begin{aligned}
 \sum_{j=1}^N w_{ij} q_j &= \frac{1}{N} \sum_{j=1}^N \sum_{k=1}^N k q_i q_j q_j = \frac{1}{N} \sum_{j=1}^N (q_j)^2 + \sum_{k=2}^N k q_i q_j q_j \\
 &= q_i^2 + \frac{1}{N} \sum_{j=1}^N \sum_{k=2}^N k q_i q_j q_j.
 \end{aligned}$$

Hier sehen wir die i -te Komponente des Eingabemusters plus eine Summe mit $(n-1)(N-1)$ Termen. Wenn diese Summanden alle statistisch unabhängig sind, können wir die Summe durch eine normalverteilte Zufallsvariable mit Standardabweichung beschreiben

$$\frac{1}{N} \frac{(n-1)(N-1) \bar{y}}{\bar{y}} = \frac{N-1}{N}.$$

Statistische Unabhängigkeit kann beispielsweise mit unkorrelierten Zufallsmustern erreicht werden. Die Summe erzeugt dann Rauschen, das nicht störend ist, solange $N \gg n$, was bedeutet, dass die Anzahl der gelernten Muster viel kleiner bleibt als die Anzahl der Neuronen. Ist jedoch $N \approx n$, dann wird der Einfluss des Rauschens so groß wie das Muster und das Netzwerk reagiert chaotisch. Eine genauere Berechnung des Phasenübergangs ergibt $N = 0,146 n$ als kritischen Punkt. Auf unser Beispiel übertragen bedeutet das, dass für 100 Neuronen bis zu 14 Muster gespeichert werden können. Da die Muster im Beispiel jedoch stark korrelieren, liegt der kritische Wert viel niedriger, offenbar zwischen 0,04 und 0,1. Selbst ein Wert von 0,146 ist viel niedriger als die Speicherkapazität einer herkömmlichen Speicherliste (Übung 9.3 auf Seite 255).

Hopfield-Netzwerke in der dargestellten Form funktionieren nur dann gut, wenn Muster mit etwa 50 % 1-Bits gelernt werden. Sind die Bits sehr asymmetrisch verteilt, müssen die Neuronen mit einer Schwelle ausgestattet werden [Ro96]. In der Physik ist dies analog zum Anlegen eines äußeren Magnetfeldes, das ebenfalls eine Asymmetrie der Spin $1/2$ - und Spin $1/2$ -Zustände bewirkt.

9.2.3 Zusammenfassung und Ausblick

Durch seine biologische Plausibilität, das gut verstandene mathematische Modell und vor allem durch die beeindruckenden Simulationen in der Mustererkennung trug das Hopfield-Modell zu einer Welle der Begeisterung für neuronale Netze und zum Aufstieg bei der Neuroinformatik als einem wichtigen Zweig der KI. Anschließend wurden viele weitere Netzwerkmodelle entwickelt. Einerseits wurden Netzwerke ohne Rückkopplungen untersucht, da deren Dynamik deutlich einfacher zu verstehen ist als bei aktuellen Hopfield-Netzwerken. Andererseits wurde versucht, die Speicherkapazität der Netzwerke zu verbessern, worauf wir im nächsten Abschnitt eingehen werden.

2Auch der Autor war von dieser Welle erfasst, die ihn 1987 von der Physik in die KI führte.

Ein besonderes Problem vieler neuronaler Modelle zeigte sich bereits beim Hopfield-Modell. Selbst wenn eine Konvergenzgarantie besteht, ist es nicht sicher, ob das Netzwerk in einen erlernten Zustand konvergiert oder bei einem lokalen Minimum hängen bleibt. Als Versuch, dieses Problem zu lösen, wurde die Boltz-Mann-Maschine mit kontinuierlichen Aktivierungswerten und einer probabilistischen Aktualisierungsregel für ihre Netzwerkdynamik entwickelt. Mithilfe eines „Temperatur“-Parameters können wir die Menge zufälliger Zustandsänderungen variieren und so versuchen, lokale Minima zu umgehen, mit dem Ziel, ein stabiles globales Minimum zu finden. Dieser Algorithmus wird „Simulated Annealing“ genannt. Glühen ist ein Prozess der Wärmebehandlung von Metallen mit dem Ziel, das Metall fester und „stabiler“ zu machen.

Das Hopfield-Modell sucht nach einem Minimum der Energiefunktion im Raum der Aktivierungswerte. Dadurch wird das in den Gewichten gespeicherte Muster gefunden, das somit in der Energiefunktion abgebildet wird. Die Hopfield-Dynamik kann auch auf andere Energiefunktionen angewendet werden, sofern die Gewichtsmatrix symmetrisch ist und die Diagonalelemente Null sind. Dies wurde von Hopfield und Tank am Problem des Handlungsreisenden erfolgreich demonstriert [HT85, Zel94]. Die Aufgabe besteht hier darin, anhand von n Städten und ihrer Entfernungsmatrix den kürzesten Hin- und Rückweg zu finden, der jede Stadt genau einmal besucht.

9.3 Neuronales Assoziatives Gedächtnis

Ein klassischer Listenspeicher kann im einfachsten Fall eine Textdatei sein, in der Ziffernfolgen zeilenweise gespeichert werden. Wenn die Datei zeilenweise sortiert ist, kann die Suche nach einem Element auch bei sehr großen Dateien sehr schnell in logarithmischer Zeit erfolgen.

Der Listenspeicher kann jedoch auch zur Erstellung von Zuordnungen verwendet werden. Ein Telefonbuch ist beispielsweise eine Abbildung der Menge aller eingegebenen Namen auf die Menge aller Telefonnummern. Diese Zuordnung wird als einfache Tabelle implementiert, die normalerweise in einer Datenbank gespeichert wird.

Eine ähnliche Aufgabe stellt die Zugangskontrolle zu einem Gebäude mittels Gesichtserkennung dar. Hier könnten wir auch eine Datenbank verwenden, in der von jeder Person ein Foto zusammen mit dem Namen der Person und möglicherweise weiteren Daten gespeichert ist. Die Kamera am Eingang macht dann ein Foto der Person und durchsucht die Datenbank nach einem identischen Foto. Wird das Foto gefunden, ist die Person identifiziert und erhält Zutritt zum Gebäude. Allerdings würde ein Gebäude mit einem solchen Kontrollsystem nicht viele Besucher bekommen, da die Wahrscheinlichkeit, dass das aktuelle Foto genau mit dem gespeicherten Foto übereinstimmt, sehr gering ist.

In diesem Fall reicht es nicht aus, das Foto nur in einer Tabelle zu speichern. Was wir vielmehr wollen, ist ein *assoziatives Gedächtnis*, das in der Lage ist, nicht nur dem Foto den richtigen Namen zuzuordnen, sondern auch einem beliebigen aus einer potenziell unendlichen Menge „ähnlicher“ Fotos. Aus einem endlichen Satz an Trainingsdaten, nämlich den mit den Namen beschrifteten gespeicherten Fotos, soll eine Funktion zur Ähnlichkeitsfindung generiert werden. Ein einfacher Ansatz hierfür ist die in Abschn. 2.1 vorgestellte Methode des nächsten Nachbarn. 8.3. Beim Lernen werden alle Fotos einfach gespeichert.

Um diese Funktion nutzen zu können, muss in der Datenbank das Foto gefunden werden, das dem aktuellen am ähnlichsten ist. Bei einer Datenbank mit vielen hochauflösenden Fotos kann dieser Vorgang je nach verwendeter Entfernungsmetrik sehr lange Rechenzeiten und erfordern

ist daher in dieser einfachen Form nicht umsetzbar. Daher bevorzugen wir anstelle eines solchen Lazy-Algorithmus einen, der die Daten in eine Funktion überführt, die dann bei ihrer Anwendung eine sehr schnelle Assoziation herstellt.

Ein weiteres Problem stellt die Suche nach einer geeigneten Distanzmetrik dar. Wir möchten, dass eine Person auch dann erkannt wird, wenn das Gesicht der Person an einer anderen Stelle auf dem Foto erscheint (Übersetzung), oder wenn es kleiner, größer oder sogar gedreht ist. Auch der Betrachtungswinkel und die Beleuchtung können variieren.

Hier zeigen neuronale Netze ihre Stärken. Ohne dass sich der Entwickler Gedanken über eine geeignete Ähnlichkeitsmetrik machen muss, liefern sie dennoch gute Ergebnisse. Wir stellen zwei der einfachsten assoziativen Gedächtnismodelle vor und beginnen mit einem Modell von Teuvo Kohonen, einem der Pioniere auf diesem Gebiet.

Das im vorherigen Kapitel vorgestellte Hopfield-Modell wäre aus zwei Gründen zu schwierig anzuwenden. Erstens handelt es sich nur um ein *autoassoziatives Gedächtnis*, also um eine annähernd identische Abbildung, die ähnliche Objekte auf das erlernte Original abbildet. Zweitens ist die komplexe wiederkehrende Dynamik in der Praxis oft schwer zu bewältigen. Daher betrachten wir nun einfache zweischichtige Feedforward-Netzwerke.

9.3.1 Korrelationsmatrixspeicher

In [Koh72] führte Kohonen ein assoziatives Gedächtnismodell ein, das auf elementarer linearer Algebra basiert. Dadurch werden Abfragevektoren $\mathbf{x} \in \mathbb{R}^N$ auf Ergebnisvektoren $\mathbf{y} \in \mathbb{R}^m$ abgebildet. Wir suchen nach einer Matrix W , die einen Satz von Trainingsdaten korrekt abbildet

$$T = \{(\mathbf{q}^1, \mathbf{t}^1), \dots, (\mathbf{q}^N, \mathbf{t}^N)\}$$

mit N Anfrage-Antwort-Paaren alle Anfragevektoren zu ihren Antworten.³ Das heißt, für $p = 1, \dots, N$ muss es so sein

$$\mathbf{t}^p = W \cdot \mathbf{q}^p, \quad (9.3)$$

oder

$$t_i^p = \sum_{j=1}^N w_{ij} q_j^p. \quad (9.4)$$

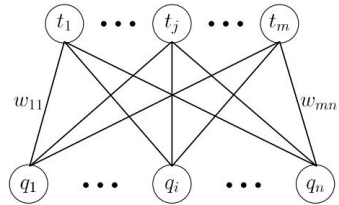
Um die Matrixelemente w_{ij} zu berechnen, gilt die Regel

$$w_{ij} = \sum_{p=1}^N t_i^p q_j^p \quad (9.5)$$

³Um eine klare Unterscheidung zwischen Trainingsdaten und anderen Werten eines Neurons zu ermöglichen, bezeichnen wir in der folgenden Diskussion den Abfragevektor als $\mathbf{q}$ und die gewünschte Antwort als $\mathbf{t}$ (Ziel).

Abb. 9.9 Darstellung des Kohonen-

Assoziativgedächtnisses als zweischichtiges neuronales Netzwerk



wird eingesetzt. Diese beiden linearen Gleichungen können einfach als neuronales Netzwerk verstanden werden, wenn wir, wie in Abb. 9.9, ein zweischichtiges Netzwerk mit q als

Eingangsschicht und t als Ausgangsschicht definieren. Die Neuronen der Ausgangsschicht

haben eine lineare Aktivierungsfunktion und als Lernregel wird (9.5) verwendet, was genau der Hebb-Regel entspricht.

Bevor wir zeigen, dass das Netzwerk die Trainingsdaten erkennt, benötigen wir folgende Definition:

Definition 9.1 Zwei Vektoren x und y heißen *orthonormal* wenn

$$x \cdot y = \begin{cases} 1, & \text{wenn } x = y, \\ 0, & \text{sonst.} \end{cases}$$

Daher

Satz 9.1 Wenn alle N Abfragevektoren q^p in den Trainingsdaten orthonormal sind, dann wird jeder Vektor q^p durch Multiplikation mit der Matrix w_{ij} aus (9.5) auf Seite 233 auf den Zielvektor t^p abgebildet.

Beweis Wir setzen (9.5) auf Seite 233 in (9.4) auf Seite 233 ein und erhalten

$$\begin{aligned} (W \cdot q^p)_i &= \sum_{j=1}^N w_{ij} q_j^p = \sum_{j=1}^N \sum_{r=1}^N r_{q_j^p} t_i^r q_j^p \\ &= \sum_{j=1}^N q_j^p q_j^{pp} t_i^p + \sum_{j=1}^N \sum_{r=p}^N q_j^p q_j^r t_i^r \\ &= t_i^p \sum_{j=1}^N q_j^p q_j^p + \sum_{r=p}^N t_i^r \sum_{j=1}^N q_j^p q_j^r \\ &= t_i^p \underbrace{(Q^p)^T p^p}_{=1} + \sum_{r=p}^N t_i^r \underbrace{(Q^r)^T p^p}_{=0} q = t_i^p \end{aligned}$$

Wenn also die Abfragevektoren orthonormal sind, werden alle Muster korrekt den jeweiligen Zielen zugeordnet. Allerdings ist die Orthonormalität eine zu starke Einschränkung. In Abschn. In Abschnitt 9.4 werden wir einen Ansatz vorstellen, der diese Einschränkung überwindet.

Da lineare Abbildungen kontinuierlich und injektiv sind, wissen wir, dass die Abbildung von Abfragevektoren auf Zielvektoren die Ähnlichkeit bewahrt. Aufgrund der Kontinuität werden ähnliche Abfragen somit auf ähnliche Ziele abgebildet. Gleichzeitig wissen wir jedoch, dass unterschiedliche Abfragen unterschiedlichen Zielen zugeordnet werden. Wenn das Netzwerk darauf trainiert wurde, Gesichtern Namen zuzuordnen, und wenn einem Gesicht der Name Henry zugewiesen wird, dann sind wir sicher, dass bei der Eingabe eines ähnlichen Gesichts eine Ausgabe ähnlich „Henry“ erzeugt wird, aber „Henry“ selbst wird garantiert nicht berechnet. Wenn die Ausgabe als String interpretiert werden kann, könnte es sich beispielsweise um „Genry“ oder „Gfnry“ handeln. Um zu dem ähnlichsten erlernten Fall zu gelangen, verwendet Kohonen eine binäre Kodierung für das Ausgabeneuron. Das berechnete Ergebnis einer Abfrage wird gerundet, wenn sein Wert nicht Null oder Eins ist. Selbst dann haben wir keine Garantie, dass wir den Zielvektor erreichen. Alternativ könnten wir eine nachträgliche Abbildung der berechneten Antwort auf den gelernten Zielvektor mit dem kleinsten Abstand h

9.3.2 Die Pseudoinverse

Es gibt einen anderen Ansatz, den wir zur Berechnung der Gewichtsmatrix W wählen können: indem wir uns alle Abfragevektoren als Spalten einer $n \times N$ -Matrix $Q = (q_1, \dots, q_N)$ und $\mathbb{R}^n$ vorstellen. Analog können die Zielvektoren als Spalten einer $m \times N$ -Matrix $T = (t_1, \dots, t_N)$ dargestellt werden. Somit kann (9.3) auf Seite 233 in die Form gebracht werden

$$T = W \cdot Q. \quad (9.6)$$

Jetzt versuchen wir, diese Gleichung nach W zu lösen. Formal invertieren wir und erhalten

$$W = T \cdot Q^{-1}. \quad (9.7)$$

Voraussetzung für diese Umrechnung ist die Invertierbarkeit von Q . Dazu muss die Matrix quadratisch sein und aus linear unabhängigen Spaltenvektoren bestehen. Das heißt, es muss gelten, dass $n = N$ ist und die n Abfragevektoren alle linear unabhängig sein müssen. Diese Bedingung ist zwar unbequem, aber immer noch nicht ganz so streng wie die oben geforderte Orthonormalität. Das Lernen nach der Hebb-Regel hat jedoch den Vorteil, dass wir sie auch dann einfach anwenden können, wenn die Abfragevektoren nicht orthonormal sind, in der Hoffnung, dass der Assoziator trotzdem einigermaßen funktioniert. Hier ist dies jedoch nicht so einfach, denn wie können wir eine nicht invertierbare Matrix umkehren?

Eine Matrix Q ist genau dann invertierbar, wenn es eine Matrix Q^{-1} mit der Eigenschaft gibt

$$Q \cdot Q^{-1} = I, \quad (9.8)$$

wobei I die Identitätsmatrix ist. Wenn Q nicht invertierbar ist (z. B. weil Q nicht quadratisch ist), dann gibt es keine Matrix Q^{-1} mit dieser Eigenschaft. Es gibt jedoch sicherlich eine Matrix, die dieser Eigenschaft nahe kommt. In diesem Sinne definieren wir

q^3			1	1	1						
q^2		1	1					1			
q^1			1			1	1				
	1	1						1		1	
		1			1	1		1			
	1	1	1	1	1	1		1	1	1	
			1	1	1					1	

$t^1 \ t^2 \ t^3$

Die Gewichtsmatrix ist somit ebenfalls binär und ein Matricelement w_{ij} ist gleich eins, wenn und nur wenn mindestens einer der Einträge $q_j^{1111}, \dots, q_j^{nnn}$ es nicht Null ist. Alle anderen Matrix Elemente sind Null. Wir sind versucht zu glauben, dass hier beim Lernen viele Informationen verloren gehen, denn wenn ein Matricelement einmal den Wert 1 annimmt, kann es durch weitere Muster nicht mehr verändert werden. Abbildung 9.10 zeigt, wie die Matrix für ein Beispiel mit $n = 10$, $m = 6$ nach dem Lernen von drei Paaren mit Einsen gefüllt wird.

[illegible]

Es ist offensichtlich, dass ein solcher Speicher weniger schnell „voll“ wird, wenn die Abfrage- und Zielvektoren nur spärlich mit Einsen gefüllt sind. Nicht nur deshalb ist die Kodierung von Ein- und Ausgabe bei assoziativen Speichern – wie auch bei anderen neuronalen Netzen – für eine gute Leistung sehr wichtig. Wir werden dies nun an einer Anwendung dieses Speichers mit geeignet gewählten Kodierungen von Ein- und Ausgabe demonstrieren.

9.3.4 Ein Rechtschreibkorrekturprogramm

Als Anwendung des beschriebenen assoziativen Gedächtnisses mit der binären Hebb-Regel wählen wir ein Programm, das fehlerhafte Eingaben korrigiert und sie auf gespeicherte Wörter aus einem Wörterbuch abbildet. Hier wäre eindeutig ein autoassoziatives Gedächtnis erforderlich. Da wir die Abfrage- und Zielvektoren jedoch unterschiedlich kodieren, ist dies nicht der Fall. Für die Abfragevektoren q wählen wir eine Paarkodierung. Für ein Alphabet mit 26 Zeichen gibt es $26 \cdot 26 = 676$ geordnete Buchstabenpaare. Bei 676 Bit hat der Abfragevektor für jedes der möglichen Paare ein Bit

aa, ab,..., az, ba,..., bz,..., za,..., zz.

Kommt in dem Wort ein Buchstabenpaar vor, so wird an der entsprechenden Stelle eine Eins eingetragen. Für das Wort „Henry“ beispielsweise sind die Plätze für „ha“, „an“ und „ns“ mit Einsen gefüllt. Für den Zielvektor t sind für jede Position im Wort bis zu einer maximalen Länge (z. B. zehn Zeichen) 26 Bit reserviert. Für den i -ten Buchstaben im Alphabet an Position j im Wort wird dann die Bitzahl $(j - 1) \cdot 26 + i$ gesetzt. Für das Wort „Henry“ sind die Bits 8, 27, 66 und 97 gesetzt. Bei maximal 10 Buchstaben pro Wort hat der Zielvektor somit eine Länge von 260 Bit.

Die Gewichtsmatrix W hat somit eine Größe von $676 \cdot 260$ Bits = 199420 Bits, was durch (9.11) auf Seite 237 kann höchstens speichern

$$N_{\max} \approx 0,69 = 0,69 \cdot \frac{676 \cdot 260}{130 \cdot 676 + 260 \cdot m + n}$$

Wörter. Bei 72 Vornamen sparen wir etwa die Hälfte ein und testen das System. Die gespeicherten Namen und die Ausgabe des Programms für mehrere Beispieleingaben sind in Abb. 9.12 auf Seite 239 dargestellt. Der Schwellenwert wird immer auf die Anzahl der Bits in der codierten Abfrage initialisiert. Hier ist dies die Anzahl der Buchstabenpaare, also die Wortlänge minus eins. Dann wird es schrittweise auf zwei reduziert. Wir könnten die Auswahl des Schwellenwerts weiter automatisieren, indem wir für jeden versuchten Schwellenwert einen Vergleich mit dem Wörterbuch durchführen und das gefundene Wort ausgeben, wenn der Vergleich erfolgreich ist.

Interessant ist die Reaktion auf die mehrdeutigen Eingaben „andr“ und „johanne“. In beiden Fällen erstellt das Netzwerk eine Mischung aus zwei gespeicherten Wörtern, die passen. Wir sehen hier eine wichtige Stärke neuronaler Netze. Sie sind in der Lage, Assoziationen zu ähnlichen Objekten ohne explizite Ähnlichkeitsmetrik herzustellen. Allerdings gibt es, ähnlich wie bei der heuristischen Suche und der menschlichen Entscheidungsfindung, keine Garantie für eine „richtige“ Lösung.

Da für alle bisher vorgestellten neuronalen Modelle die Trainingsdaten in Form von Input-Output-Paaren vorliegen müssen, handelt es sich um überwachtes Lernen, was auch für die in den folgenden Abschnitten vorgestellten Netzwerke der Fall ist.

Gespeicherte Wörter: agathe, agnes, alexander, andreas, andree, anna, annemarie, astrid, august, bernhard, björn, cathrin, christian, christoph, corinna, corrado, dieter, elisabeth, elvira, erdmüt, ernst, evelyn, fabrizio, frank, franz, geoffrey, georg, gerhard, hannelore, harry, herbert, ingilt, irmgard, jan, johannes, johnny, juergen, karin, klaus, ludwig, luise, manfred, maria, mark, markus, marleen, martin, matthias, norbert, otto, Patricia, Peter, Phillip, Quit, Reinhold, Renate, Robert, Robin, Sabine, Sebastian, Stefan, Stephan, Sylvie, Ulrich, Ulrike, Ute, Uwe, Werner, Wolfgang, Xavier	
Verbände des Programms:	
Eingabemuster: Harry Schwellenwert: 4, Antwort: Harry Schwellenwert: 3, Antwort: Harry Schwellenwert: 2, Antwort: horryrde ----- Eingabemuster: ute Schwelle: 2, Antwort: ute ----- Eingabemuster: Gerhar Schwelle: 5, Antwort: Gerhard Schwelle: 4, Antwort: Gerrarn Schwelle: 3, Antwort: Jerrhrd Schwelle: 2, Antwort: Jurtyrde ----- Eingabemuster: egrhard Schwelle: 6, Antwort: Schwelle: 5, Antwort: Schwelle: 4, Antwort: Gerhard Schwelle: 3, Antwort: gernhrd Schwelle: 2, Antwort: irrryrde ----- Eingabemuster: andr Schwellenwert: 3, Antwort: Andrees Schwellenwert: 2, Antwort: Anexenser	Eingabemuster: andrees Schwellenwert: 6, Antwort: a Schwellenwert: 5, Antwort: Andree Schwellenwert: 4, Antwort: Andrees Schwellenwert: 3, Antwort: Mnnrens Schwellenwert: 2, Antwort: Morxsnsr ----- Eingabemuster: johanne Schwellenwert: 6, Antwort: johnnnes Schwellenwert: 5, Antwort: johnnnes Schwellenwert: 4, Antwort: jornnnrse Schwellenwert: 3, Antwort: sorrrysrse Schwellenwert: 2, Antwort: wtrrsyrse ----- Eingabemuster: johnnnes Schwelle: 6, Antwort: joh Schwelle: 5, Antwort: johnnnes Schwelle: 4, Antwort: johnnyes Schwelle: 3, Antwort: jonnnyes Schwelle: 2, Antwort: jornsyrse ----- Eingabemuster: johnnyes Schwellenwert: 7, Antwort: Schwellenwert: 6, Antwort: Joh Schwellenwert: 5, Antwort: Johnny Schwellenwert: 4, Antwort: Johnnyes Schwellenwert: 3, Antwort: Johnnyes Schwellenwert: 2, Antwort: Jonnnyes

Abb. 9.12 Anwendung des Rechtschreibkorrekturprogramms auf verschiedene gelernte oder fehlerhafte Eingaben. Die richtigen Eingaben werden mit dem maximalen (d. h. dem ersten Versuch) Schwellenwert gefunden. Bei fehlerhaften Eingaben muss für eine korrekte Zuordnung die Schwelle gesenkt werden

9.4 Lineare Netzwerke mit minimalen Fehlern

Die in den bisher vorgestellten neuronalen Modellen verwendete Hebb-Regel arbeitet mit Assoziationen zwischen benachbarten Neuronen. Im assoziativen Gedächtnis wird dies ausgenutzt, um eine Zuordnung von Abfragevektoren zu Zielen zu lernen. Dies funktioniert in vielen Fällen sehr gut, insbesondere wenn die Abfragevektoren linear unabhängig sind. Ist diese Bedingung nicht erfüllt, beispielsweise weil zu viele Trainingsdaten vorliegen, stellt sich die Frage: Wie finden wir die optimale Gewichtsmatrix? Optimal bedeutet, dass der durchschnittliche Fehler minimiert wird.

Wir Menschen sind in der Lage, aus Fehlern zu lernen. Die Hebb-Regel bietet diese Möglichkeit nicht. Der im Folgenden beschriebene Backpropagation-Algorithmus verwendet eine elegante, aus der Funktionsnäherung bekannte Lösung, um die Gewichte so zu ändern, dass der Fehler in den Trainingsdaten minimiert wird.

Es seien N Paare von Trainingsvektoren

$$T = \{(q^1, t^1), \dots, (q^N, t^N)\}$$

gegeben sein mit $q^p \in [0, 1]^N$, $t^p \in [0, 1]^M$. Wir suchen eine Funktion $f: [0, 1]^N \rightarrow [0, 1]^M$, die den quadratischen Fehler minimiert

$$\sum_{p=1}^N (f(q^p) - t^p)^2$$

auf die Daten. Nehmen wir zunächst an, dass die Daten keine Widersprüche enthalten. Das heißt, es gibt keinen Abfragevektor in den Trainingsdaten, der zwei verschiedenen Zielen zugeordnet werden sollte. In diesem Fall ist es nicht schwierig, eine Funktion zu finden, die den quadratischen Fehler minimiert. Tatsächlich gibt es unendlich viele Funktionen, die den Fehler auf Null bringen. Wir definieren die Funktion

$$f(q) = 0, \text{ wenn } q \notin \{q^1, \dots, q^N\}$$

Und

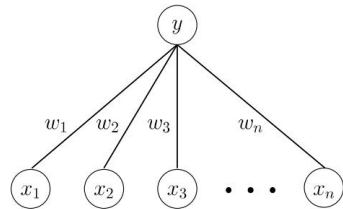
$$f(q^p) = t^p \quad \forall p \in \{1, \dots, N\}.$$

Dies ist eine Funktion, die sogar den Fehler in den Trainingsdaten auf Null setzt. Was wollen wir mehr? Warum sind wir mit dieser Funktion nicht zufrieden?

Die Antwort lautet: Weil wir ein intelligentes System bauen wollen! Intelligent bedeutet unter anderem, dass sich die gelernte Funktion gut von den Trainingsdaten auf neue, unbekannte Daten aus demselben repräsentativen Datensatz verallgemeinern lässt. Mit anderen Worten bedeutet es: Wir wollen keine Überanpassung der Daten durch Auswendiglernen. Was wollen wir denn wirklich?

Wir wollen eine Funktion, die glatt ist und den Raum zwischen den Punkten „ausgleicht“. Kontinuität und die Fähigkeit, mehrere Derivate zu übernehmen, wären sinnvolle Anforderungen. Da es auch unter diesen Bedingungen noch unendlich viele Funktionen gibt, die den Fehler zu Null machen, müssen wir diese Klasse von Funktionen noch weiter einschränken.

Abb. 9.13 Ein zweischichtiges Netzwerk mit einem Ausgangs-Neuron



9.4.1 Methode der kleinsten Quadrate

Die einfachste Wahl ist eine lineare Abbildung. Wir beginnen mit einem zweischichtigen Netzwerk (Abb. 9.13), in dem das einzelne Neuron der zweiten Schicht seine Aktivierung anhand berechnet

$$y = f \sum_{i=1}^N w_i x_i$$

mit $f(x) = x$. Die Tatsache, dass wir hier nur Output-Neuronen betrachten, stellt keine wirkliche Einschränkung dar, da ein zweischichtiges Netzwerk mit zwei oder mehr Output-Neuronen immer in unabhängige Netzwerke mit identischen Input-Neuronen für jedes der ursprünglichen Output-Neuronen aufgeteilt werden kann. Die Gewichte der Teilnetze sind alle unabhängig. Die Verwendung einer Sigmoidfunktion anstelle der linearen Aktivierung bringt hier keinen Vorteil, da die Sigmoidfunktion streng monoton wachsend ist und die Ordnungsbeziehung zwischen verschiedenen Ausgabewerten nicht v

Gewünscht ist ein Vektor w , der den quadratischen Fehler minimiert

$$E(w) = \sum_{p=1}^N (w^T \tilde{y}^p - t^p)^2 = \sum_{p=1}^N \sum_{i=1}^N w_i \tilde{y}_i^p t^p$$

Als notwendige Bedingung für ein Minimum dieser Fehlerfunktion müssen alle partiellen Ableitungen Null sein. Wir fordern also für $j = 1, \dots, n$:

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^N \sum_{i=1}^n w_i \tilde{y}_i^p t^p \tilde{y}_j^p = 0.$$

Multipliziert man dies, erhält man ein Ergebnis

$$\sum_{p=1}^N \sum_{i=1}^n w_i \tilde{y}_i^p \tilde{y}_j^p t^p = 0,$$

und das Vertauschen der Summen ergibt das lineare Gleichungssystem

$$\sum_{i=1}^N w_{ij} \sum_{p=1}^P q_i^p q_j^p = \sum_{p=1}^P t_p q_j^p,$$

welches mit

$$A_{ij} = \sum_{p=1}^P q_i^p q_j^p \quad \text{und} \quad b_j = \sum_{p=1}^P t_p q_j^p \quad (9.12)$$

kann als Matrixgleichung geschrieben werden

$$Aw = b. \quad (9.13)$$

Diese sogenannten Normalgleichungen haben immer mindestens eine Lösung und, wenn A invertierbar ist, genau eine. Darüber hinaus ist die Matrix A positiv-definit, was zur Folge hat, dass die gefundene Lösung im Einzelfall ein globales Minimum ist.

Dieser Algorithmus ist als Methode der kleinsten Quadrate bekannt.

Die Rechenzeit für den Aufbau der Matrix A wächst mit $\tilde{y}(N \cdot n^2)$ und die Zeit zum Lösen des Gleichungssystems als $O(n^3)$. Diese Methode kann sehr einfach auf die Einbeziehung mehrerer Ausgabeneuronen erweitert werden, da, wie bereits erwähnt, bei zweischichtigen Feedforward-Netzwerken die Ausgabeneuronen unabhängig voneinander sind.

9.4.2 Anwendung auf die Appendizitis-Daten

Als Anwendung ermitteln wir nun einen linearen Score für die Blinddarmentzündungsdiagnose. Aus den aus Abschn. 8.4.5 verwenden wir die Methode der kleinsten Quadrate, um eine lineare Abbildung von Symptomen auf die kontinuierlichen Klassenvariablen *AppScore* mit Werten im Intervall $[0, 1]$ zu bestimmen und die lineare Kombination zu erhalten

$$\begin{aligned} \text{AppScore} = & 0,00085 \text{Alter} + 0,125 \text{ Geschlecht} + 0,025 \text{P1Q} + 0,035 \text{P2Q} + 0,021 \text{P3Q} \\ & + 0,025 \text{P4Q} + 0,12 \text{ TensLoc} + 0,031 \text{ TensGlo} + 0,13 \text{ Losl} \\ & + 0,081 \text{Conv} + 0,0034 \text{RectS} + 0,0027 \text{ TAxI} + 0,0031 \text{ TRec} \\ & + 0,000021 \text{ Leuko} + 0,11 \text{Diab} + 1,83. \end{aligned}$$

Diese Funktion liefert kontinuierliche Variablen für *AppScore*, obwohl die eigentliche binäre Klassenvariable *App* nur die Werte 0 und 1 annimmt. Wir müssen uns also wie beim Perzeptron für einen Schwellenwert entscheiden. Der Klassifizierungsfehler des Scores als Funktion des Schwellenwerts ist in Abb. 9.14 auf Seite 243 für die Trainingsdaten und die Testdaten aufgeführt. Wir sehen deutlich, dass beide Kurven nahezu gleich sind und haben

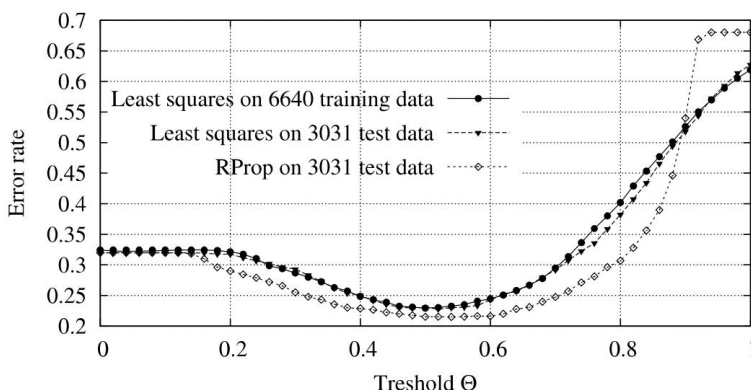


Abb. 9.14 Fehler der kleinsten Quadrate für Trainings- und Testdaten

ihr Minimum bei $\bar{y} = 0,5$. An der geringen Differenz der beiden Kurven erkennen wir, dass eine Überanpassung für diese Methode kein Problem darstellt, da das Modell sehr gut aus den Testdaten verallgemeinert.

Ebenfalls in der Abbildung dargestellt ist das Ergebnis für das nichtlineare, dreischichtige RProp-Netzwerk (Abschn. 9.5) mit einem etwas geringeren Fehler für Schwellenwerte zwischen 0,2 und 0,9. Für die praktische Anwendung des abgeleiteten Scores und die korrekte Bestimmung des Schwellenwerts $\bar{y}$ ist es wichtig, nicht nur den Fehler zu betrachten, sondern auch nach Fehlerart (nämlich falsch positiv und falsch negativ) zu unterscheiden, wie dies in der LEXMED-Anwendung geschieht in Abb. 7.10 auf Seite 142. In der dort gezeigten ROC-Kurve ist auch der hier berechnete Score dargestellt. Wir sehen, dass das einfache lineare Modell dem LEXMED-System deutlich unterlegen ist. Offensichtlich sind lineare Approximationen für viele komplexe Anwendungen nicht leistungstark genug.

9.4.3 Die Delta-Regel

Die Methode der kleinsten Quadrate ist, wie das Perzeptron- und Entscheidungsbaum-Lernen, ein sogenannter *Batch-Learning-Algorithmus*, im Gegensatz zum *inkrementellen Lernen*. Beim Batch-Lernen müssen alle Trainingsdaten in einem Durchlauf gelernt werden. Wenn neue Trainingsdaten hinzukommen, können diese nicht einfach zusätzlich zu den bereits vorhandenen gelernt werden. Der gesamte Lernvorgang muss mit dem erweiterten Satz wiederholt werden. Dieses Problem wird durch inkrementelle Lernalgorithmen gelöst, die das gelernte Modell an jedes weitere neue Beispiel anpassen können. In den Algorithmen, die wir in der folgenden Diskussion betrachten werden, werden wir die Gewichte für jedes neue Trainingsbeispiel durch die Regel addi-

$$w_j = w_j + \Delta w_j .$$

Um eine inkrementelle Variante der Methode der kleinsten Quadrate abzuleiten, betrachten wir die oben berechneten n partiellen Ableitungen der Fehlerfunktion noch einmal

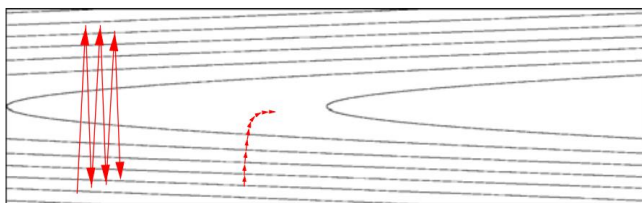


Abb. 9.15 Gradientenabstieg für ein großes η (links) und ein sehr kleines η (rechts) in ein Tal, das flach nach rechts abfällt. Für ein großes η gibt es Schwingungen um das Tal herum. Bei einem zu kleinen η hingegen erfolgt die Konvergenz im flachen Tal sehr langsam

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^N \sum_{i=1}^N w_{ip} \tilde{y}_i t_p - p_{qj}$$

Der Farbverlauf

$$\vec{\partial E} = \left(\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right)$$

als Vektor aller partiellen Ableitungen der Fehlerfunktion zeigt in die Richtung des stärksten Anstiegs der Fehlerfunktion im n -dimensionalen Raum der Gewichte. Bei der Suche nach einem Minimum folgen wir daher der Richtung des negativen Gradienten. Als Formel zur Änderung der Gewichte erhalten wir

$$\tilde{y} w_j = \tilde{y} \left(\frac{\partial E}{\partial w_j} \right) = \tilde{y} \left(2 \sum_{p=1}^N \sum_{i=1}^N w_{ip} \tilde{y}_i t_p - p_{qj} \right),$$

wobei die *Lernrate* $\tilde{y}$ eine frei wählbare positive Konstante ist. Ein größeres $\tilde{y}$ beschleunigt die Konvergenz, erhöht aber gleichzeitig das Risiko einer Oszillation um Minima oder flache Täler. Daher ist die optimale Wahl von $\tilde{y}$ keine einfache Aufgabe (siehe Abb. 9.15).

Am Anfang wird oft ein großes $\tilde{y}$ verwendet, zum Beispiel $\tilde{y} = 1$, das dann langsam schrumpft.

Durch Ersetzen der Aktivierung

$$p_y = \sum_{i=1}^N p_{wiq i}$$

des Ausgabeneurons für das angewandte Trainingsbeispiel die Formel ist vereinfacht und erhalten wir die *Delta-Regel*

$$w_j = \tilde{y} \sum_{p=1}^N (t_p \tilde{y}_y - p_y) q_p$$

Abb. 9.16 Lernen eines zweischichtigen linearen Netzwerks mit der Delta-Regel. Beachten Sie, dass die Gewichtsänderungen immer dann auftreten, nachdem alle Trainingsdaten angewendet wurden

DELTALEARNING(*TrainingExamples*, $\tilde{y}$)

Initialisieren Sie alle Gewichte w_j zufällig

Wiederholen

Sie $w = 0$

Für alle $(q, p, t_p) \in \text{Trainingsbeispiele}$

Berechnen Sie die $P = w p q p$

Netzwerkausgabe $y_w = w +$

$\tilde{y}(t_p - y_p) q p w = w + w$

Bis w konvergiert

DELTALEARNINGINCREMENTAL(*TrainingExamples*, $\tilde{y}$)

Initialisieren Sie alle Gewichte w_j zufällig

Wiederholen

Für alle $(q, p, t_p) \in \text{Trainingsbeispiele}$

Berechnen Sie die $P = w p q p$

Netzwerkausgabe $y_w = w + \tilde{y}(t_p - y_p) q p$

Bis w konvergiert

Abb. 9.17 Inkrementelle Variante der Delta-Regel

Somit wird für jedes Trainingsbeispiel die Differenz zwischen dem Ziel t_p und dem tatsächlichen p des Netzwerks für die gegebene Eingabe q berechnet. Nach der Summierung aller Muster über die Ausgabe y werden die Gewichte dann proportional zur Summe geändert. Dieser Algorithmus ist in Abb. 9.16 dargestellt.

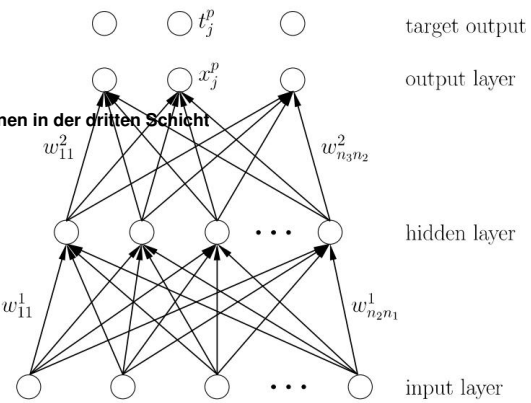
Wir sehen, dass der Algorithmus immer noch nicht wirklich inkrementell ist, da die Gewichtsänderungen erst auftreten, nachdem alle Trainingsbeispiele einmal angewendet wurden. Diesen Mangel können wir beheben, indem wir nach jedem Trainingsbeispiel direkt die Gewichte ändern (inkrementeller Gradient des Scent) (Abb. 9.17), was streng genommen keine korrekte Umsetzung der Delta-Regel mehr darstellt.

9.4.4 Vergleich zum Perzeptron

Die in Abschn. eingeführte Lernregel für Perzeptrene. 8.2.1, die Methode der kleinsten Quadrate und die Delta-Regel können verwendet werden, um lineare Funktionen aus Daten zu generieren. Bei Perzeptronen wird jedoch im Gegensatz zu den anderen Methoden ein Klassifikator für linear trennbare Klassen durch die Schwellenwertentscheidung gelernt.

Die anderen beiden Methoden erzeugen jedoch eine lineare Näherung der Daten. Wie in Abschn. 9.4.2 kann aus der linearen Abbildung bei Bedarf durch Anwendung einer Schwellenwertfunktion

Abb. 9.18 Ein dreischichtiges Backpropagation-Netzwerk mit n_1 Neuronen in der ersten, n_2 Neuronen in der zweiten und n_3 Neuronen in der dritten Schicht



Das Perzeptron und die Delta-Regel sind iterative Algorithmen, bei denen die Zeit bis zur Konvergenz stark von den Daten abhängt. Bei linear trennbaren Daten lässt sich für das Perzeptron eine Obergrenze für die Anzahl der Iterationsschritte finden. Für die Delta-Regel hingegen gibt es nur eine Garantie der asymptotischen Konvergenz ohne Limit [HKP91].

Bei der Methode der kleinsten Quadrate besteht das Lernen darin, ein lineares Gleichungssystem für den Gewichtsvektor aufzustellen und zu lösen. Es gibt also eine harte Grenze für die Rechenzeit. Aus diesem Grund ist die Methode der kleinsten Quadrate immer dann vorzuziehen, wenn kein inkrementelles Lernen erforderlich ist.

9.5 Der Backpropagation-Algorithmus

Mit dem Backpropagation-Algorithmus stellen wir nun das am häufigsten verwendete neuronale Modell vor. Der Grund für seine weite Verbreitung ist seine universelle Vielseitigkeit für beliebige Approximationsaufgaben. Der Algorithmus geht direkt auf die inkrementelle Delta-Regel zurück. Im Gegensatz zur Delta-Regel wendet sie als Aktivierungsfunktion eine nichtlineare Sigmoidfunktion auf die gewichtete Summe der Eingaben an. Darüber hinaus kann ein Backpropagation-Netzwerk mehr als zwei Neuronenschichten haben. Bekannt wurde der Algorithmus durch den Artikel [RHR86] in der legendären PDP-Sammlung [RM86].

In Abb. 9.18 ist ein typisches Backpropagation-Netzwerk mit einer Eingabeschicht und einer verborgenen und eine Ausgabebene wird angezeigt. Da der aktuelle Ausgabewert x der P Ausgabeschicht dargestellt Neuronenschicht j mit dem Zielausgabewert t verglichen wird, werden diese parallel zu j gezeichnet. Mit Ausnahme der Eingabeneuronen berechnen alle Neuronen ihren aktuellen Wert x_j nach der Regel

$$x_j = f \left(\sum_{i=1}^N w_{ji} x_i \right) \tag{9.14}$$

Dabei ist n die Anzahl der Neuronen in der vorherigen Schicht. Wir verwenden die Sigmoidfunktion

$$f(x) = \frac{1}{1 + e^{-x}}$$

Analog zur inkrementellen Delta-Regel werden die Gewichte proportional zum negativen Gradienten der über die Ausgabeneuronen summierten quadratischen Fehlerfunktion geändert

$$E_p(w) = 2 \sum_{\text{Ausgabe}} \frac{1}{k} (t_j - y_j)^2$$

für das Trainingsmuster p :

$$\Delta w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

Zur Ableitung der Lernregel wird E_p durch den obigen Ausdruck ersetzt. Innerhalb des Ausdrucks wird x_k durch (9.14) auf Seite 246 ersetzt. Innerhalb der Gleichung erfolgen die Ausgaben x_i der Neuronen der nächsten, tieferen Schicht rekursiv usw. Durch mehrfache Anwendung der Kettenregel (siehe [RHR86] bzw [Zel94]) erhalten wir die Backpropagation-Lernregel

$$\Delta w_{ji} = -\eta \delta_j x_i$$

mit

$$\delta_j = \sum_k w_{kj} \delta_k \quad \text{wenn } j \text{ ein Ausgabeneuron ist,} \\ \delta_j = -\sum_k w_{kj} \delta_k \quad \text{wenn } j \text{ ein verstecktes Neuron ist,}$$

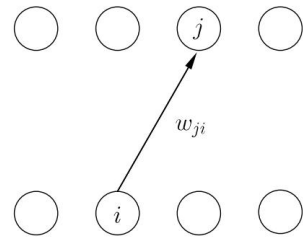
was auch als verallgemeinerte Delta-Regel bezeichnet wird. Für alle Neuronen enthält die Formel zur Gewichtsänderung Δw_{ji} von Neuron i zu Neuron j (siehe Abb. 9.19 auf Seite 248) wie die Hebb-Regel einen Term $\delta_j x_i$. Der neue Faktor $(1 - y_j)$ erzeugt das Symmetrietry, der in der Hebb-Regel fehlt, zwischen den Aktivierungen 0 und 1 des Neurons. Für die

Ausgabeneuronen der Faktor $(1 - y_j)$ kümmert sich um einen Gewichtsveränderungs-Profi proportional zum Fehler. Für die verborgenen Neuronen der δ_j des Neurons j wird berechnet Wert δ_j rekursiv aus allen δ_k der Neuronen der nächsthöheren Ebene.

Änderungen δ_j Der gesamte Ablauf des Lernprozesses ist in Abb. 9.20 auf Seite 248 dargestellt.

Nach der Berechnung der Ausgabe des Netzwerks (Vorwärtsausbreitung) für ein Trainingsbeispiel wird der Approximationsfehler berechnet. Dies wird dann während der Rückwärtsausbreitung verwendet, um die Gewichte von Schicht zu Schicht rückwärts zu ändern. Der gesamte Vorgang wird dann auf alle Trainingsbeispiele angewendet und wiederholt, bis sich die Gewichte nicht mehr ändern oder ein Zeitlimit erreicht ist.

Abb. 9.19 Bezeichnung der Neuronen und Gewichte für die Anwendung der Backpropagation-Regel



BACKPROPAGATION(TrainingExamples, $\tilde{y}$)

Initialisieren Sie alle Gewichte w_j auf Zufallswerte.

Wiederholen Sie den Vorgang für alle (q, p, t)

p auf die Eingabeschicht 1. $p \in \tilde{y}$ Trainingsbeispiele

Wenden Sie den Abfragevektor

q an 2. Vorwärtsausbreitung: Für alle Schichten ab der ersten verborgenen Schicht aufwärts Für jedes Neuron der Schicht

Berechnen Sie die Aktivierung $x_j = \sum_{i=1}^n w_{ji} x_i$

3. Berechnung des quadratischen Fehlers $E_p(w)$

4. Rückwärtsausbreitung: Für

alle Gewichtungsebenen von der letzten abwärts. Für

jedes Gewicht $w_{ji} = w_{ji}$

$$w_{ji} = w_{ji} + \tilde{y}_p - \sum_i x_i^p$$

Bis w konvergiert oder das Zeitlimit erreicht ist

Abb. 9.20 Der Backpropagation-Algorithmus

Wenn wir ein Netzwerk mit mindestens einer verborgenen Schicht aufbauen, können nichtlineare Abbildungen gelernt werden. Ohne verborgene Schichten sind die Ausgabeneuronen trotz der Sigmoidfunktion nicht leistungsfähiger als ein lineares Neuron. Der Grund dafür ist, dass die Sigmoidfunktion streng monoton ist. Gleiches gilt für mehrschichtige Netzwerke, die lediglich eine lineare Funktion als Aktivierungsfunktion verwenden, beispielsweise die Identitätsfunktion. Dies liegt daran, dass verkettete Ausführungen linearer Zuordnungen insgesamt linear sind.

Ebenso wie beim Perzeptron die Klasse der darstellbaren Funktionen werden auch vergrößert, wenn wir eine variable Sigmoidfunktion verwenden

$$f(x) = \frac{1}{1 + e^{\tilde{y}(x - \tilde{y})}}$$

mit Schwelle $\tilde{y}$. Die Umsetzung erfolgt analog zu dem in Abschn. 8.2, bei dem in die Eingabeschicht und in jede verborgene Schicht ein Neuron eingefügt wird, dessen Aktivierung immer den Wert Eins hat und das mit Neuronen der nächsthöheren Ebene verbunden ist. Die Gewichte dieser Verbindungen werden normal gelernt und stellen den Schwellenwert $\tilde{y}$ der Nachfolgeneuronen dar.

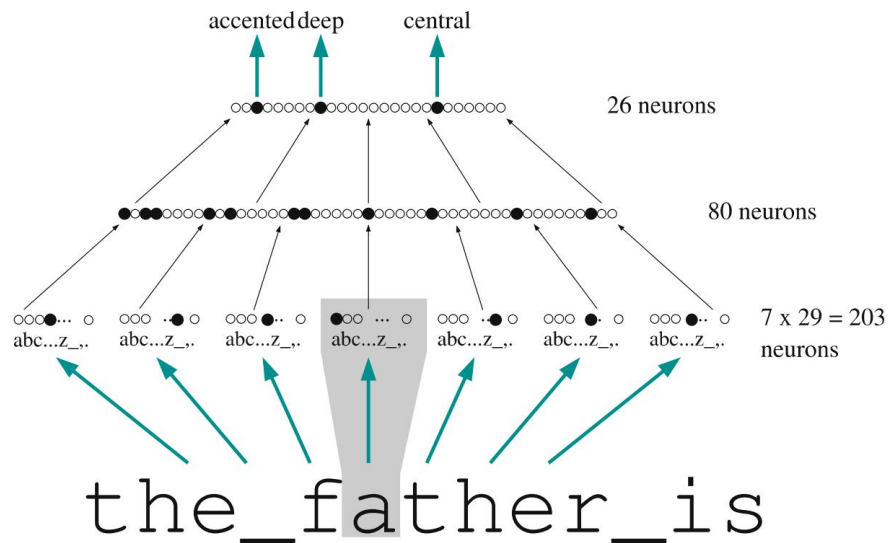


Abb. 9.21 Das NETalk-Netzwerk ordnet einen Text seinen Ausspracheattributen zu

9.5.1 NETalk: Ein Netzwerk lernt sprechen

Sejnowski und Rosenberg haben 1986 sehr eindrucksvoll gezeigt, wozu Backpropagation in der Lage ist [SR86]. Sie haben ein System entwickelt, das in der Lage ist, englischen Text aus einer Textdatei verständlich vorzulesen. Die Architektur des in Abb. 9.21 dargestellten Netzwerks besteht aus einer Eingabeschicht mit $7 \times 29 = 203$ Neuronen

Dabei werden der aktuelle Buchstabe und drei vorangegangene Buchstaben sowie drei nachfolgende Buchstaben kodiert. Für jeden dieser sieben Buchstaben sind 29 Neuronen reserviert

Zeichen „a...z „ Die Eingabe wird auf die 26 Ausgabeneuronen über 80 abgebildet

versteckte Neuronen, von denen jedes für ein bestimmtes Phonem steht. Zum Beispiel die „a“ in „Vater“ wird tief, akzentuiert und zentral ausgesprochen. Das Netzwerk wurde geschult mit 1.000 Wörtern, die zufällig Buchstabe für Buchstabe nacheinander angewendet wurden.

Für jeden Buchstaben wurde manuell die Zielausgabe für seine Intonation angegeben. Um die Intonationsattribute in tatsächliche Klänge zu übersetzen, Teil des Sprachsynthesystems

Es wurde DECtalk verwendet. Durch vollständige Interkonnektivität enthält das Netzwerk ein Ganzes von $203 \times 80 + 80 \times 26 = 18320$ Gewichte.

Das System wurde mithilfe eines Simulators auf einer VAX 780 mit etwa 50 Zyklen trainiert über alle Wörter. Bei etwa 5 Zeichen pro Wort im Durchschnitt also etwa 50-1000 =

Es waren 250.000 Iterationen des Backpropagation-Algorithmus erforderlich. Mit einer Geschwindigkeit von ca. Ein Zeichen pro Sekunde bedeutet dies etwa 69 Stunden Rechenzeit. Der

Die Entwickler beobachteten viele Eigenschaften des Systems, die denen des Menschen sehr ähnlich sind. Lernen. Das System kann zunächst nur undeutlich sprechen oder einfache Wörter verwenden. Mit der Zeit

Es verbesserte sich weiter und erreichte schließlich eine 95-prozentige Korrektheit der ausgesprochenen Buchstaben.

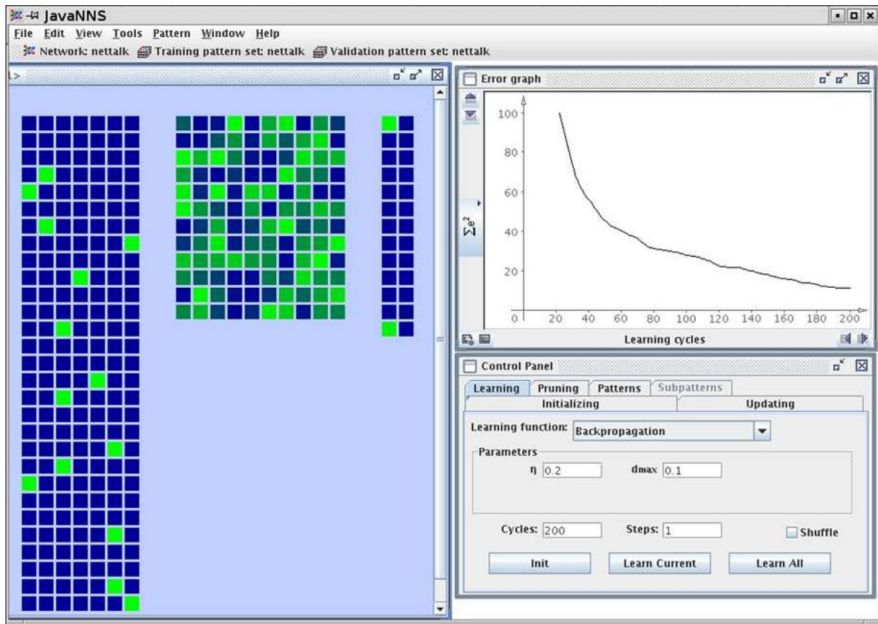


Abb. 9.22 Das NETtalk-Netzwerk in JNNS. Im *linken Fenster* sehen wir von *links nach rechts* die 7 · 29 Eingabeneuronen, 80 versteckte und 26 Ausgabeneuronen. Aufgrund ihrer großen Anzahl werden die Gewichte im Netzwerk weggelassen. *Oben rechts* ist eine Lernkurve zu sehen, die die Entwicklung des quadratischen Fehlers über die Zeit zeigt

Für visuelle Experimente mit neuronalen Netzen empfiehlt sich der Java Neural Network Simulator JNNS [Zel94]. Das mit JNNS geladene und trainierte NETtalk-Netzwerk ist in Abb. 9.22 dargestellt .

9.5.2 Erlernen von Heuristiken für Theorembeweiser

Im Kap. In Kapitel 6 wurden Algorithmen für die heuristische Suche, wie der A* -Algorithmus und der IDA* -Algorithmus, diskutiert. Um eine signifikante Reduzierung des Suchraums zu erreichen, ist eine gute Heuristik für die Implementierung dieser Algorithmen erforderlich. In Abschn. In Abschnitt 4.1 wurde das Problem des explodierenden Suchraums während der Beweissuche von Theorembeweisern aufgezeigt. Dieses Problem wird durch die große Anzahl möglicher Inferenzschritte in jedem Schritt verursacht.

Wir versuchen nun, heuristische Proof-Controlling-Module zu erstellen, die für den nächsten Schritt die verschiedenen Alternativen bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Falle einer Auflösung könnte die Bewertung der verfügbaren Klauseln durch eine Funktion erfolgen, die basierend auf bestimmten Attributen von Klauseln wie der Anzahl der Literale, der Komplexität der Begriffe usw. einen Wert für jedes Paar von Auflösungen berechnet able-Klauseln. In [ESS89, SE90] wurde eine solche Heuristik für den Theorembeweis SETHEO mittels Backpropagation erlernt. Beim Erlernen der Heuristik von

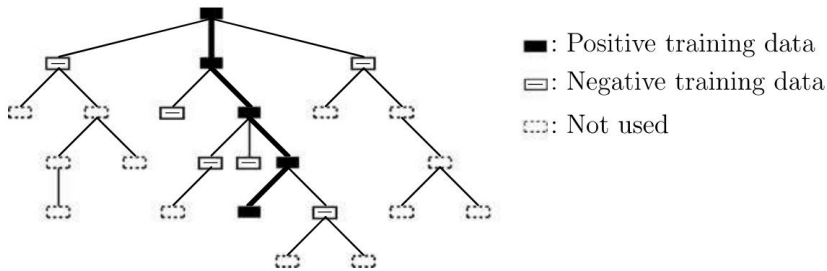


Abb. 9.23 Suchbaum mit einem erfolgreichen Pfad, dessen Knoten als positiv bewertet werden, und den negativ bewerteten erfolglosen Zweigen

Die aktuelle Klausel wurde für jeden gefundenen Beweis als Trainingsdaten für die positive Klasse gespeichert in jeder Filiale während der Suche. Bei allen Zweigen, die nicht zu einem Beweis führten, wurden die Attribute der aktuellen Klausel als Trainingsdaten für die negative Klasse gespeichert.

Es wird ein solcher Baum mit einem erfolgreichen Pfad und den entsprechenden Knotenbewertungen skizziert in Abb. 9.23.

Auf diesen Daten wurde ein Backpropagation-Netzwerk trainiert und anschließend zur Auswertung genutzt. Klauseln im Prüfer. Für jede Klausel wurden 16 numerische Attribute berechnet, normiert und in einem Eingabeneuron kodiert. Das Netzwerk wurde mit 25 versteckten Neuronen und ein Ausgabeneuron für die Klasse (positiv/negativ).

Es zeigte sich, dass die Zahl der Schlussfolgerungsversuche um ein Vielfaches reduziert werden kann bei schwierigen Problemen um Größenordnungen, was letztendlich die Rechenzeit von Stunden auf Sekunden reduziert. Dadurch wurde es möglich, Theoreme zu beweisen, die ohne Heuristik unerreicht waren.

9.5.3 Probleme und Verbesserungen

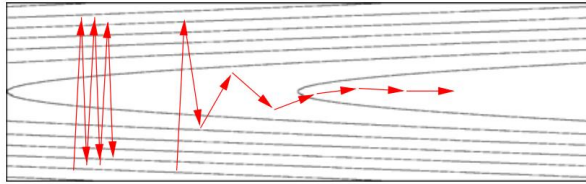
Backpropagation ist mittlerweile 25 Jahre alt und hat sich in verschiedenen Anwendungen bewährt, zum Beispiel in der Mustererkennung und in der Robotik. Allerdings ist seine Anwendung manchmal problematisch. Vor allem, wenn das Netzwerk viele tausend Gewichte hat und Da viele Trainingsdaten gelernt werden müssen, treten zwei Probleme auf:

Das Netzwerk konvergiert häufig gegen lokale Minima der Fehlerfunktion. Außerdem, Die Rückausbreitung konvergiert oft sehr langsam. Das bedeutet, dass viele Iterationen vorbei sind. Alle Trainingsmuster werden benötigt. Zur Linderung dieser Probleme wurden viele Verbesserungen vorgeschlagen. Wie in Abschn. 9.4.3 können Schwingungen vermieden werden durch langsame Reduzierung der Lernrate η , wie in Abb. 9.15 auf Seite 244 dargestellt.

Eine weitere Methode zur Reduzierung von Schwingungen ist die Verwendung eines Impulsterms. Aktualisierung der Gewichte, wodurch sichergestellt wird, dass sich die Richtung des Gradientenabfalls nicht ändert sich von einem Schritt zum nächsten zu dramatisch verändern. Hier zum aktuellen Gewicht $w_{ij}(t)$ zum Zeitpunkt t wird ein weiterer Teil der Änderung $\Delta w_{ij}(t-1)$ aus dem vorherigen Schritt hinzugefügt. Die Lernregel ändert sich dann zu

$$w_{ij}(t) = \eta \delta_j x_i + w_{ij}(t-1) + \eta \Delta w_{ij}(t-1)$$

Abb. 9.24 Abrupte Richtung
Änderungen werden durch geglättet
die Nutzung des Impulses
Begriff. Eine Iteration ohne die
Impulsterm (*links*) in
Vergleich zur Iteration mit
der Impulsterm (*rechts*)



mit einem Parameter $\bar{y}$ zwischen Null und Eins. Dies wird zweidimensional dargestellt
Beispiel in Abb. 9.24.

Eine andere Idee besteht darin, die lineare Fehlerfunktion anstelle des quadratischen Fehlers zu minimieren
Funktion, die das Problem der langsamen Konvergenz in flache Täler verringert.

Der Gradientenabstieg bei der Backpropagation basiert letztlich auf einer linearen Approximation
der Fehlerfunktion. Quickprop, ein Algorithmus, der eine quadratische Näherung an die Fehlerfunktion
verwendet und so eine schnellere Konvergenz erreicht, wurde von entwickelt
Scott Fahlmann.

Durch geschickte Vereinheitlichung der genannten Verbesserungen und weiterer heuristischer
Tricks gelang Martin Riedmiller eine weitere Optimierung mit dem RProp-Algorithmus [RB93]. RProp
hat die Backpropagation ersetzt und ist der neue Zustand von
Kunst-Feedforward-Approximationsalgorithmus für neuronale Netze. In Abschn. 8,8 haben wir angewendet
RProp hat bei der Klassifizierung der Blinddarmentzündungsdaten einen Fehler festgestellt
ungefähr die gleiche Größe wie ein erlernter Entscheidungsbaum.

9.6 Support Vector Machines

Feedforward-Neuronale Netze mit nur einer Gewichtsschicht sind linear. Linearität
führt zu einfachen Netzwerken und schnellem Lernen mit garantierter Konvergenz. Darüber hinaus ist
die Gefahr einer Überanpassung bei linearen Modellen gering. Für viele Anwendungen,
Allerdings sind die linearen Modelle nicht stark genug, etwa weil die relevanten Klassen nicht linear
trennbar sind. Hier kommen mehrschichtige Netzwerke wie Backpropagation zum Einsatz, mit der
Folge, dass lokale Minima, Konvergenz entstehen

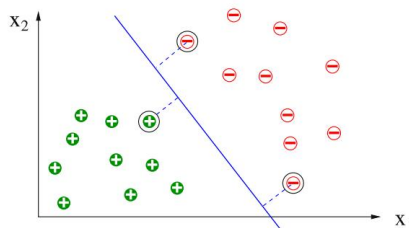
Es kann zu Problemen und einer Überanpassung kommen.

Ein vielversprechender Ansatz, der die Vorteile linearer und nichtlinearer Modelle vereint, folgt der
Theorie der Support Vector Machines (SVM), die wir vorstellen werden
Grobe Gliederung anhand eines Zwei-Klassen-Problems.⁴

Im Fall zweier linear separierbarer Klassen ist es leicht, einen dividierenden Hyper zu finden
Ebene, zum Beispiel mit der Perzeptron-Lernregel. Allerdings gibt es sie in der Regel
unendlich viele solcher Ebenen, wie im zweidimensionalen Beispiel in Abb. 9.25 auf
Seite 253. Gesucht wird eine Ebene, die zu beiden den größten Mindestabstand hat
Klassen. Diese Ebene wird in der Regel durch wenige Punkte im Randbereich eindeutig definiert.
Diese Punkte, die sogenannten *Stützvektoren*, haben alle den gleichen Abstand zur Teilung

⁴Support-Vektor-Maschinen sind keine neuronalen Netze. Aufgrund ihrer historischen Entwicklung und ihrer
mathematischen Verwandtschaft zu linearen Netzwerken werden sie hier jedoch diskutiert.

Abb. 9.25 Zwei Klassen mit der maximalen Trennlinie. Die eingekreisten Punkte sind die Stützvektoren



Linie. Um die Unterstützungsvektoren zu finden, gibt es einen effizienten Optimierungsalgorithmus. Interessant ist, dass die optimale teilende Hyperebene durch einige Parameter bestimmt wird, nämlich durch die Stützvektoren. Daher ist die Gefahr einer Überanpassung gering.

Support-Vektor-Maschinen wenden diesen Algorithmus in einem zweistufigen Prozess auf nichtlinear trennbare Probleme an: Im ersten Schritt wird eine nichtlineare Transformation auf die Daten angewendet, mit der Eigenschaft, dass die transformierten Daten linear trennbar sind. Im zweiten Schritt werden dann die Stützvektoren im transformierten Raum bestimmt.

Der erste Schritt ist hochinteressant, aber nicht ganz einfach. Tatsächlich ist es immer möglich, die Klassen durch Transformation des Vektorraums linear trennbar zu machen, solange die Daten keine Widersprüche enthalten.⁵ Eine solche Trennung kann beispielsweise durch Einführung einer neuen $(n + 1)$ -ten Dimension und erreicht werden die Definition

$$x_{n+1} = \begin{cases} 1, & \text{wenn } x \in \text{Klasse} \\ 0, & \text{wenn } x \notin \text{Klasse} \end{cases}$$

Diese Formel hilft jedoch nicht viel, da sie nicht auf neue Punkte einer unbekannten Klasse anwendbar ist, die klassifiziert werden sollen. Wir benötigen daher eine allgemeine Transformation, die möglichst unabhängig von den aktuellen Daten ist. Es kann gezeigt werden, dass es solche generischen Transformationen auch für beliebig geformte Klassenteilungsgrenzen im ursprünglichen Vektorraum gibt. Im transformierten Raum sind die Daten dann linear trennbar. Allerdings wächst die Anzahl der Dimensionen des neuen Vektorraums exponentiell mit der Anzahl der Dimensionen des ursprünglichen Vektorraums.

Allerdings ist die Vielzahl der neuen Dimensionen nicht so problematisch, da bei der Verwendung von Stützvektoren die Teilungsebene, wie oben erwähnt, nur durch wenige Parameter bestimmt wird.

Die zentrale nichtlineare Transformation des Vektorraums wird als *Kernel bezeichnet*, weshalb Support-Vektor-Maschinen auch als Kernel-Methoden bezeichnet werden. Die ursprünglich für Klassifikationsaufgaben entwickelte SVM-Theorie wurde erweitert und kann nun auch auf Regressionsprobleme angewendet werden.

Die hier verwendete Mathematik ist sehr interessant, aber für eine erste Einführung zu umfangreich. Um tiefer in diesen vielversprechenden jungen Zweig des maschinellen Lernens einzutauchen, verweisen wir den Leser auf [SS02, Alp04] und [Bur98].

⁵Ein Datenpunkt ist widersprüchlich, wenn er zu beiden Klassen gehört.

9.7 Anwendungen

Neben den hier aufgeführten Beispielen gibt es unzählige Anwendungen für neuronale Netze in allen Industriezweigen. Ein sehr wichtiger Bereich ist die Mustererkennung in all ihren Formen, sei es die Analyse von Fotos zur Erkennung von Personen oder Gesichtern, die Erkennung von Fischeschwärmen anhand von Sonarbildern, die Erkennung und Klassifizierung von Militärfahrzeugen anhand von Radarscans und vieles mehr. Aber auch neuronale Netze können darauf trainiert werden, gesprochene Sprache und handgeschriebenen Text zu erkennen.

Neuronale Netze dienen nicht nur der Erkennung von Objekten und Szenen. Sie werden außerdem darauf trainiert, einfache Roboter mithilfe von Sensordaten zu steuern sowie die Suche in Backgammon- und Schachcomputern heuristisch zu steuern. Besonders interessant in Spielen sind Reinforcement-Learning-Techniken Abschn. 10.8, zum Beispiel mit neuronalen Netzen.

Neuronale Netze werden neben statistischen Methoden seit langem erfolgreich zur Prognose von Aktienmärkten und zur Beurteilung der Bonität von Bankkunden eingesetzt.

Für viele dieser Anwendungen können auch andere Algorithmen des maschinellen Lernens verwendet werden. Allerdings sind neuronale Netze bislang erfolgreicher und beliebter als alle anderen Algorithmen des maschinellen Lernens. Aufgrund des großen kommerziellen Erfolgs von Data Mining und Support-Vektor-Maschinen wird die historische Verbreitung neuronaler Netze jedoch langsam zugunsten anderer Methoden zurückgedrängt.

9.8 Zusammenfassung und Ausblick

Mit dem Perzeptron, der Delta-Regel und der Backpropagation haben wir die wichtigste Klasse neuronaler Netze und ihre Beziehung zu Scores und zum naiven Bayes einerseits, aber auch zur Methode der kleinsten Quadrate eingeführt. Als nächstes sahen wir die faszinierenden Hopfield-Netzwerke, inspiriert von biologischen Modellen, die jedoch aufgrund ihrer komplexen Dynamik in der Praxis schwer zu handhaben sind. Von größerer Bedeutung für die Praxis sind die verschiedenen assoziativen Gedächtnismodelle.

In allen neuronalen Modellen werden Informationen über viele Gewichte verteilt gespeichert. Aus diesem Grund hat das Absterben einiger Neuronen im Gehirn keinen spürbaren Einfluss auf die Gehirnfunktion. Durch die verteilte Datenspeicherung ist das Netzwerk robust gegenüber kleinen Störungen. Dies hängt auch mit der Fähigkeit zusammen, Muster mit Fehlern zu erkennen, die in mehreren Beispielen gezeigt werden.

Die verteilte Darstellung von Wissen hat jedoch den Nachteil, dass es für den Wissensingenieur schwierig ist, Informationen zu lokalisieren. Es ist praktisch unmöglich, die vielen Gewichte in einem vollständig trainierten neuronalen Netzwerk zu analysieren und zu verstehen. Bei einem erlernten Entscheidungsbaum hingegen ist es einfach, das erlernte Wissen zu verstehen und sogar als logische Formel darzustellen. Besonders ausdrucksstark und elegant ist die Prädikatenlogik, die die Formalisierung von Beziehungen ermöglicht. Beispielsweise ist das Prädikat „Großmutter“ (karen, clyde) leicht zu verstehen. Auch mit neuronalen Netzen lässt sich ein solcher Zusammenhang erlernen, allerdings ist es nicht einmal möglich, das „Großmutterneuron“ im Netz zu lokalisieren. Daher gibt es immer noch Probleme bei der Verbindung neuronaler Netze mit Symbolverarbeitungssystemen.

Viele der vielen interessanten neuronalen Modelle können hier nicht behandelt werden. Beispielsweise handelt es sich bei den von Kohonen eingeführten selbstorganisierenden Karten um eine biologisch motivierte Abbildung von einer Sensorschicht von Neuronen auf eine zweite Schicht von Neuronen mit der Eigenschaft, dass diese Abbildung adaptiv ist und die Ähnlichkeit bewahrt.

Ein Problem mit den hier vorgestellten Netzwerken tritt beim inkrementellen Lernen auf. Wenn beispielsweise ein vollständig trainiertes Backpropagation-Netzwerk mit neuen Mustern weiter trainiert wird, geraten viele (und möglicherweise alle) der alten Muster schnell in Vergessenheit. Um dieses Problem zu lösen, entwickelten Carpenter und Grossberg die adaptive Resonanztheorie (ART), die zu einer ganzen Reihe neuronaler Modelle führte.

Als zusätzliche Literatur empfehlen wir die Lehrbücher [Bis05, RMS92, Roj96, Zel94]. Wer sich für die Lektüre des wichtigsten Originalwerks auf diesem spannenden Gebiet interessiert, kann die beiden Sammelbände [AR88, APR90] konsultieren.

9.9 Übungen

9.9.1 Von der Biologie zur Simulation

Aufgabe 9.1 Zeigen Sie die Punktsymmetrie der Sigmoidfunktion zu $(\bar{y}, \frac{1}{2})$.

9.9.2 Hopfield-Netzwerke

Übung 9.2 Verwenden Sie das Hopfield-Netzwerk-Applet unter <http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html> und testen Sie die Speicherkapazität für korrelierte und unkorrelierte Muster (mit zufällig gesetzten Bits) mit einem 10×10 Gittergröße. Verwenden Sie zum Testen ein Muster mit 10 % Rauschen.

Aufgabe 9.3 Vergleichen Sie den theoretischen Grenzwert $N = 0,146 n$ für die maximale Anzahl speicherbarer Muster aus Abschn. 9.2.2, mit der Kapazität eines klassischen Binärspeichers gleicher Größe.

9.9.3 Lineare Netzwerke mit minimalen Fehlern

Übung 9.4

- (a) Schreiben Sie ein Programm zum Erlernen einer linearen Abbildung mit der Methode der kleinsten mittleren Quadrate. Das ist ganz einfach: Man muss nur die Normalgleichungen nach (9.12) auf Seite 242 aufstellen und dann das Gleichungssystem lösen.
- (a) Wenden Sie dieses Programm auf die Blinddarmentzündungsdaten auf der Website dieses Buches an und ermitteln Sie einen linearen Wert. Geben Sie den Fehler als Funktion des Schwellenwerts an, wie in Abb. 9.14 auf Seite 243. (c) Bestimmen Sie nun die ROC-Kurve für diesen Score.

9.9.4 Backpropagation

Übung 9.5 Erstellen Sie mit einem Backpropagation-Programm (zum Beispiel in JNNS oder KNIME) ein Netzwerk mit acht Eingabeneuronen, acht Ausgabeneuronen und drei versteckten Neuronen. Trainieren Sie dann das Netzwerk mit acht Trainingsdatenpaaren q , so dass im p -ten pp mit der Requisite q Vektor q das p -te Bit eins und alle anderen Bits null sind. Das Netzwerk hat somit die einfache Aufgabe, eine identische Abbildung zu lernen. Ein solches Netzwerk wird als 8-3-8-Encoder bezeichnet. Beobachten Sie nach dem Lernprozess die Kodierung der drei verborgenen Neuronen für die Eingabe aller acht gelernten Eingabevektoren. Was fällt dir auf? Wiederholen Sie nun das Experiment und reduzieren Sie die Anzahl der verborgenen Neuronen auf zwei und dann auf eins.

Übung 9.6 Um zu zeigen, dass Backpropagation-Netzwerke nicht linear trennbare Mengen teilen können, trainieren Sie die XOR-Funktion. Die Trainingsdaten hierfür sind: $((0, 0), 0)$, $((0, 1), 1)$, $((1, 0), 1)$, $((1, 1), 0)$. (a) Erstellen Sie ein Netzwerk mit jeweils zwei Neuronen in der Eingabeschicht und der verborgenen Schicht, und ein Neuron in der Ausgabeschicht und trainieren dieses Netzwerk. (b) Löschen Sie nun die verborgenen Neuronen und verbinden Sie die Eingabeschicht direkt mit den Ausgabeneuronen. Was beobachten Sie?

Aufgabe 9.7 (a)

Zeigen Sie, dass jedes mehrschichtige Backpropagation-Netzwerk mit einer linearen Aktivierungsfunktion genauso leistungsfähig ist wie ein zweischichtiges. Dazu reicht es zu zeigen, dass es sich bei aufeinanderfolgenden Ausführungen linearer Abbildungen um eine lineare Abbildung handelt. (b) Zeigen Sie, dass ein zweischichtiges Backpropagation-Netzwerk mit einer streng monotonen Aktivierungsfunktion für Klassifizierungsaufgaben nicht leistungsfähiger ist als eines ohne Aktivierungsfunktion oder mit einer linearen Aktivierungsfunktion.

9.9.5 Support Vector Machines

Übung 9.8 Zwei nicht linear trennbare zweidimensionale Sätze von Trainingsdaten M_+ $2 \times 2 = 1$ und alle $+$ und M_- sind gegeben. Alle Punkte in M_+ liegen innerhalb des Einheitskreises $x_1^2 + x_2^2 \leq 1$ in $\mathbb{R}^2$, was ergibt Punkte in M_- liegen außerhalb. Geben Sie eine Koordinatentransformation $f: \mathbb{R}^2 \rightarrow \mathbb{R}^n$, um die Daten linear trennbar zu machen. Geben Sie die Gleichung der Trennlinie an und skizzieren Sie die beiden Räume und die Datenpunkte.

Verstärkungslernen

10

10.1 Einführung

Alle bisher beschriebenen Lernalgorithmen – mit Ausnahme der Clustering-Algorithmen – gehören zur Klasse des *überwachten Lernens*. Beim überwachten Lernen soll der Agent eine Zuordnung von den Eingabevariablen zu den Ausgabevariablen lernen. Dabei ist es wichtig, dass für jedes einzelne Trainingsbeispiel alle Werte sowohl für Eingabevariablen als auch für Ausgabevariablen bereitgestellt werden. Mit anderen Worten: Wir benötigen einen Lehrer oder eine Datenbank, in der die zu erlernende Abbildung für eine ausreichende Anzahl von Eingabewerten näherungsweise definiert ist. Die einzige Aufgabe des maschinellen Lernalgorithmus besteht darin, das Rauschen aus den Daten herauszufiltern und eine Funktion zu finden, die die Zuordnung auch zwischen den gegebenen Datenpunkten gut annähert.

Beim Reinforcement Learning ist die Situation anders und schwieriger, da keine Trainingsdaten verfügbar sind. Wir beginnen mit einem einfachen Anschauungsbeispiel aus der Robotik, das dann als Anwendung für die verschiedenen Algorithmen dient.

Reinforcement Learning ist im Bereich der Robotik sehr wertvoll, wo die auszuführenden Aufgaben häufig komplex genug sind, um sich der Kodierung als Programme zu entziehen, und wo keine Trainingsdaten verfügbar sind. Die Aufgabe des Roboters besteht darin, durch Versuch und Irrtum (oder Erfolg) herauszufinden, welche Handlungen in einer bestimmten Situation *gut* sind und welche nicht. In vielen Fällen lernen wir Menschen auf ganz ähnliche Weise. Wenn ein Kind beispielsweise laufen lernt, geschieht dies meist ohne Anleitung, sondern einfach durch Verstärkung. Erfolgreiche Gehversuche werden mit Fortschritten belohnt, erfolglose Gehversuche werden mit oft schmerzhaften Stürzen bestraft. Positive und negative Verstärkung sind auch wichtige Faktoren für erfolgreiches Lernen in der Schule und in vielen Sportarten (siehe Abb. 10.1 auf Seite 258).

Im folgenden Beispiel wird eine stark vereinfachte Bewegungsaufgabe erlernt.

Beispiel 10.1 Ein Roboter, dessen Mechanismus nur aus einem rechteckigen Block und einem Arm mit zwei Gelenken gy und gx besteht, ist in Abb. 10.2 auf Seite 258 dargestellt (siehe [KMK97]). Die einzigen möglichen Aktionen des Roboters sind die Bewegung von gy nach oben oder unten und von gx nach rechts oder links. Darüber hinaus erlauben wir nur Bewegungen fester diskreter Einheiten (z. B. in 10-Grad-Schritten). Die Aufgabe des Agenten besteht darin, eine *Richtlinie*



Abb. 10.1 „Vielleicht sollte ich das nächste Mal etwas früher mit dem Abbiegen beginnen oder langsamer fahren?“ – Lernen aus negativer Verstärkung

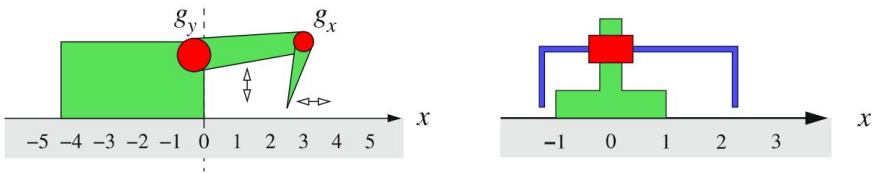


Abb. 10.2 Durch die Bewegung seiner beiden Gelenke kann sich der Krabbelroboter links *im* Bild vorwärts und rückwärts bewegen. Der Laufroboter auf der *rechten Seite* muss den Rahmen entsprechend *nach oben* und *unten* bzw. *nach links* und *rechts* bewegen. Das Feedback für die Bewegung des Roboters ist bei einer Bewegung nach rechts positiv *und* bei einer Bewegung nach links negativ

Dadurch kann es sich möglichst schnell nach rechts bewegen. Der Laufroboter in Abb. 10.2 arbeitet analog innerhalb desselben zweidimensionalen Zustandsraums.

Eine erfolgreiche Aktionssequenz ist in Tabelle 10.1 auf Seite 259 dargestellt. Die Aktion zum Zeitpunkt $t = 2$ führt dazu, dass der belastete Arm den Körper um eine Längeneinheit nach rechts bewegt. Schöne Animationen dieses Beispiels finden Sie in [KMK97] und [Tok06].

Bevor wir auf den Lernalgorithmus eingehen, müssen wir die Aufgabe mathematisch passend modellieren. Wir beschreiben den *Zustand* des Roboters durch die beiden Variablen g_x und g_y für die Position der Gelenke mit jeweils endlich vielen diskreten Werten. Der Zustand des Roboters wird somit als Vektor (g_x, g_y) kodiert. Die Anzahl der möglichen Gelenkpositionen beträgt n_x bzw. n_y . Zur Bewertung der Aktionen des Roboters nutzen wir die horizontale Position des Roboterkörpers, die reale Werte annehmen kann. Bewegungen nach rechts werden

oder

Tabelle 10.1 Ein Zyklus einer periodischen Reihe von Bewegungen mit systematischer Vorwärtsbewegung

Kriechender Roboter	Laufender Roboter	Zeit τ	Zustand		Belohnungsaktion	
			g_y	g_x	x	bei
		0	Hoch	Links	0	Rechts
		1	Hoch	Rechts	0	Runter
		2	Unten rechts		0	Links
		3	Unten links1			Hoch

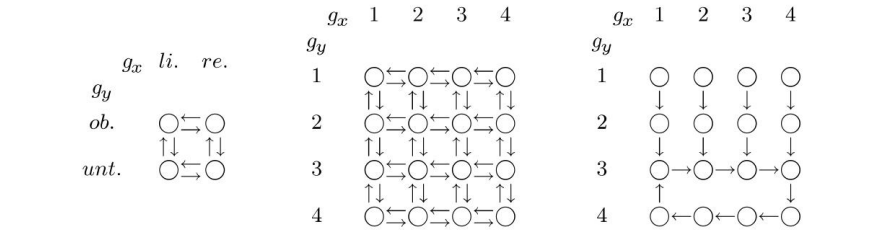


Abb. 10.3 Der Zustandsraum des Beispielroboters bei jeweils zwei möglichen Positionen die Gelenke (links) und bei jeweils vier horizontalen und vertikalen Positionen (Mitte). Im Mit dem richtigen Bild ist eine optimale Politik gegeben

mit positiven Änderungen an x und Bewegungen nach links werden mit negativen bestraft Änderungen.

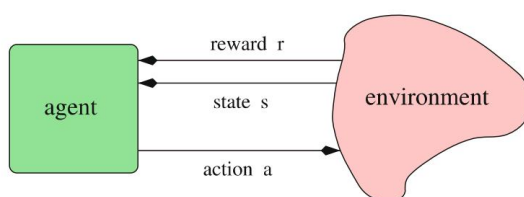
In Abb. 10.3 ist der Zustandsraum für zwei Varianten davon vereinfacht dargestellt.1 Im linken Beispiel haben beide Gelenke zwei Positionen, im mittleren Beispiel beide jeder hat vier Positionen. Eine optimale Richtlinie ist in Abb. 10.3 rechts dargestellt.

10.2 Die Aufgabe

Wie in Abb. 10.4 auf Seite 260 dargestellt, unterscheiden wir zwischen dem Agenten und seiner Umgebung. Zum Zeitpunkt t ist die Welt, die den Agenten und seine Umgebung umfasst, vorhanden beschrieben durch einen Zustand $s_t \in S$. Die Menge S ist eine Abstraktion der tatsächlich möglichen Zustände der Welt, weil einerseits die Welt nicht genau beschrieben werden kann, und andererseits Andererseits verfügt der Agent oft nur über unvollständige Informationen über die tatsächliche Situation

1Der aus Bögen bestehende Armbewegungsraum wird als rechtwinkliges Gitter dargestellt.

Abb. 10.4 Der Agent und seine Interaktion mit der Umgebung



Zustand aufgrund von Messfehlern. Der Agent führt dann zum Zeitpunkt t eine **Aktion** bei $\tilde{y}_A$ aus. Diese Aktion verändert die Welt und führt somit zum Zustand $st+1$ zum Zeitpunkt $t+1$. Die durch die Umgebung definierte **Zustandsübergangsfunktion** $\tilde{y}$ bestimmt den neuen Zustand $st+1 = \tilde{y}(st, at)$. Es kann vom Agenten nicht beeinflusst werden.

Nach Ausführung der Aktion at erhält der Agent eine sofortige Belohnung $rt = r(st, at)$ (siehe Abb. 10.4). Die unmittelbare Belohnung $rt = r(st, at)$ ist immer abhängig vom aktuellen Zustand und der ausgeführten Aktion. $r(st, at) = 0$ bedeutet, dass der Agent zum Zeitpunkt keine unmittelbare Rückmeldung für die Aktion erhält. Während des Lernens sollte $rt > 0$ zu einer positiven und $rt < 0$ zu einer negativen Verstärkung der Bewertung der Aktion im Zustand st führen. Insbesondere beim Reinforcement Learning werden Anwendungen untersucht, bei denen es über einen längeren Zeitraum zu keiner unmittelbaren Belohnung kommt. Ein Schachspieler beispielsweise lernt, sein Spiel durch gewonnene oder verlorene Partien zu verbessern, auch wenn er in einzelnen Zügen keine sofortige Belohnung für alle erhält. Hier zeigt sich die Schwierigkeit, die Belohnung am Ende einer Handlungsfolge allen Handlungen in der Folge zuzuordnen, die zu diesem Punkt geführt haben (Credit-Assignment-Problem).

Beim kriechenden Roboter besteht der Zustand aus der Stellung der beiden Gelenke, das heißt, $s = (gx, gy)$. Die Belohnung ergibt sich aus der zurückgelegten Strecke x .

Eine **Richtlinie** $\tilde{y} : S \rightarrow A$ ist eine Abbildung von Zuständen auf Aktionen. Das Ziel des Verstärkungslernens besteht darin, dass der Agent auf der Grundlage seiner Erfahrungen eine optimale Richtlinie findet. Eine Richtlinie ist optimal, wenn sie langfristig, also über viele Schritte hinweg, den Nutzen maximiert. Aber was bedeutet „Belohnung maximieren“ genau? Wir definieren den **Wert** bzw. die **ermäßigte Belohnung**

$$V^{\tilde{y}}(st) = r_t + \gamma V^{\tilde{y}}(st+1) \quad \text{für } t=0, 1, 2, \dots \quad (10.1)$$

einer Richtlinie $\tilde{y}$, wenn wir im Startzustand st beginnen. Dabei ist $0 < \gamma < 1$ eine Konstante, die sicherstellt, dass zukünftiges Feedback umso mehr abgezinst wird, je weiter es in der Zukunft liegt. Die unmittelbare Belohnung rt wird am stärksten gewichtet. Diese Belohnungsfunktion wird am häufigsten verwendet. Eine manchmal interessante Alternative ist die durchschnittliche Belohnung

$$V^{\tilde{y}}(st) = \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i} \quad (10.2)$$

Eine Politik $\tilde{y}$ heißt **optimal**, wenn für alle Zustände s

$$V^{\tilde{y}}(s) \geq V^{\tilde{y}'}(s) \quad (10.3)$$

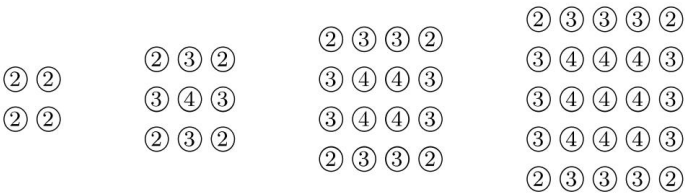


Abb. 10.5 Der Zustandsraum für das Beispiel mit den Werten 2, 3, 4, 5 für n_x und n_y . Die Anzahl der möglichen Aktionen ist für jedes Bundesland in den jeweiligen Kreisen angegeben

Tabelle 10.2 Anzahl der Richtlinien für unterschiedlich große Zustandsräume im Beispiel

n_x, n_y	Anzahl der Staaten	Anzahl der Policen
$24 \cdot 24 = 16$		
$39 \cdot 243 \cdot 44 = 5184$		
4	16	$243 \cdot 84 \cdot 4 \approx 2,7 \times 10^7$
5	25	$243 \cdot 124 \cdot 9 \approx 2,2 \times 10^{12}$

Das heißt, sie ist hinsichtlich des definierten Wertes mindestens so gut wie alle anderen Policen Funktion. Zur besseren Lesbarkeit ist die optimale Wertfunktion V^* mit V bezeichnet.

Die hier besprochenen Agenten bzw. deren Richtlinien verwenden nur Informationen über den aktuellen Zustand, um den nächsten Zustand zu bestimmen, und nicht die Vorgeschichte. Dies ist gerechtfertigt, wenn die Belohnung einer Aktion nur vom aktuellen Zustand und den aktuellen Aktionen abhängt. Solche Prozesse werden **Markov-Entscheidungsprozesse** (MDP) genannt. In vielen Anwendungen, insbesondere in der Robotik, ist der tatsächliche Zustand des Agenten nicht genau bekannt, was die Planung von Aktionen zusätzlich erschwert. Der Grund dafür kann ein verrauschtes Sensorsignal sein. Wir nennen einen solchen Prozess einen **teilweise beobachtbaren Markov-Entscheidungsprozess** (POMDP).

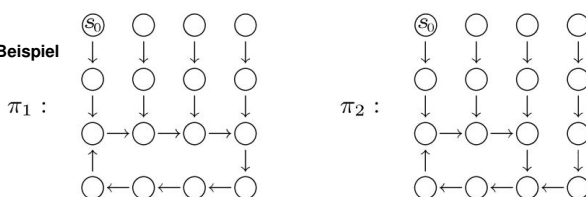
10.3 Uninformierte kombinatorische Suche

Die einfachste Möglichkeit, eine erfolgreiche Politik zu finden, ist die kombinatorische Aufzählung aller Politiken, wie in Kap. 6. Allerdings gibt es bereits im einfachen Beispiel 10.1 sehr viele Richtlinien, was dazu führt, dass die kombinatorische Suche mit einem extrem hohen Rechenaufwand verbunden ist. In Abb. 10.5 ist für jeden Zustand die Anzahl der möglichen Aktionen angegeben. Daraus wird die Anzahl der möglichen Richtlinien als Produkt der angegebenen Werte berechnet, wie in Tabelle 10.2 dargestellt.

Für beliebige Werte von n_x und n_y gibt es immer vier Eckknoten mit zwei möglichen Aktionen, $2(n_x - 2) + 2(n_y - 2)$ Randknoten mit drei Aktionen und $(n_x - 2)(n_y - 2)$ innere Knoten mit vier Aktionen. So gibt es

$$4 \cdot 2 \cdot 3 \cdot 2(n_x - 2) + 2(n_y - 2) + 4(n_x - 2)(n_y - 2)$$

Abb. 10.6 Zwei unterschiedliche Richtlinien für das Beispiel



Unterschiedliche Richtlinien für feste n_x und n_y . Die Zahl der Policen wächst also exponentiell mit der Zahl der Staaten. Dies gilt im Allgemeinen, wenn es mehr als eine mögliche Aktion pro Zustand gibt. Für praktische Anwendungen ist dieser Algorithmus daher unbrauchbar. Sogar die heuristische Suche, beschrieben in Kap. 6, kann hier nicht verwendet werden. Da die direkte Belohnung für fast alle Aktionen Null ist, kann sie nicht als heuristische Bewertungsfunktion verwendet werden.

Der Rechenaufwand steigt sogar noch höher, wenn man bedenkt, dass (zusätzlich zu (s) für jedes Aufzählung aller Richtlinien), der Wert V ⁹ generierte (s) berechnet werden muss, für das Politik $\tilde{y}$ und jeder Startzustand s . Die unendliche Summe in V ⁹ abgeschnitten werden muss Verwendung in einer praktischen Berechnung; Aufgrund der exponentiellen Reduzierung der $\tilde{y}$ - Faktoren in (10.1) auf Seite 260 stellt dies jedoch kein Problem dar.

In Beispiel 10.1 auf Seite 257 kann die Differenz $x_{t+1} - \tilde{y}_t$ als unmittelbare Belohnung für eine Aktion bei verwendet werden, was bedeutet, dass jede Bewegung des Roboterkörpers nach rechts mit 1 und jede Bewegung des Roboterkörpers nach links mit -1 belohnt wird. Die linke wird mit $\tilde{y}_1$ bestraft. In Abb. 10.6 sind zwei Richtlinien dargestellt. Hier ist die unmittelbare Belohnung überall außer in der unteren Reihe des Zustandsraums Null. Die linke Politik $\tilde{y}_1$ ist langfristig besser, da bei langen Aktionssequenzen der durchschnittliche Fortschritt pro Aktion $3/8 = 0,375$ für $\tilde{y}_1$ und $2/6 \approx 0,333$ für $\tilde{y}_2$ beträgt. Wenn wir (10.1) (s) verwenden, ist das Ergebnis die folgende Tabelle mit dem auf Seite 260 für V Startzustand s_0 am

oben links und verschiedene $\tilde{y}$ -Werte:

$\bar{y}$	0,9	0,8375	0,8	
$V \bar{y}_1 (s_0)$	2,52	1,156	$V \bar{y}_2 (s_0)$	0,77
	2,39	1,156		0,80

Hier sehen wir, dass die Richtlinie $\tilde{y}_1$ der Richtlinie $\tilde{y}_2$ überlegen ist, wenn $\Gamma = 0,9$, und das Gegenteil gilt, wenn $\Gamma = 0,8$. Für $\tilde{y} = 0,8375$ sind beide Richtlinien gleich gut.

Wir können deutlich erkennen, dass ein größeres $\bar{y}$ zu einem größeren Zeithorizont für die Bewertung von Richtlinien führt.

10.4 Wertiteration und dynamische Programmierung

Bei dem naiven Ansatz, alle Richtlinien aufzuzählen, wird viel redundante Arbeit geleistet, da viele Richtlinien größtenteils identisch sind. Sie dürfen sich nur geringfügig unterscheiden. Dennoch wird jede Police komplett neu erstellt und bewertet.

Dies legt nahe, Zwischenergebnisse für Teile von Richtlinien zu speichern und wiederzuverwenden.

Dieser Ansatz zur Lösung von Optimierungsproblemen wurde bereits 1957 von Richard Bellman als *dynamische Programmierung* eingeführt [Bel57]. Bellman erkannte das Für eine optimale Politik gilt:

Unabhängig vom Startzustand s_t und der ersten Aktion bei müssen alle Folgeentscheidungen ausgehend von jedem möglichen Nachfolgezustand s_{t+1} optimal sein.

Basierend auf dem sogenannten Bellman-Prinzip wird es möglich, eine globale Lösung zu finden optimale Politik durch lokale Optimierung einzelner Maßnahmen. Wir werden dies ableiten Prinzip für MDPs zusammen mit einem geeigneten Iterationsalgorithmus.

Gewünscht ist eine optimale Politik $\tilde{v}$ was (10.3) auf Seite 260 und (10.1) auf Seite 260 erfüllt Seite 260. Wir schreiben die beiden Gleichungen um und erhalten

$$V^{\tilde{v}}(s_t) = \max_{\text{bei } t=1, \text{ bei } t=2, \dots} (r(s_t, a_t) + \tilde{\gamma} r(s_{t+1}, a_{t+1}) + \tilde{\gamma}^2 r(s_{t+2}, a_{t+2}) + \dots). \quad (10.4)$$

Da die unmittelbare Belohnung $r(s_t, a_t)$ nur von s_t und a_t abhängt, nicht aber von der Nachfolgezustände und -aktionen können maximiert werden, was letztendlich der Fall ist ergibt die folgende rekursive Charakterisierung von $V^{\tilde{v}}$:

$$\begin{aligned} V^{\tilde{v}}(s_t) &= \max_{\text{bei}} [r(s_t, a_t) + \tilde{\gamma} \max_{\text{bei } t=1, \text{ bei } t=2, \dots} (r(s_{t+1}, a_{t+1}) + \tilde{\gamma} r(s_{t+2}, a_{t+2}) + \dots)] \\ &= \max_{\text{bei}} [r(s_t, a_t) + \tilde{\gamma} V^{\tilde{v}}(s_{t+1})]. \end{aligned} \quad (10.5)$$

Gleichung (10.5) ergibt sich aus der Substitution $t \rightarrow t+1$ in (10.4). Etwas geschrieben einfacher:

$$V^{\tilde{v}}(s) = \max_A [r(s, a) + \tilde{\gamma} V^{\tilde{v}}(\tilde{y}(s, a))]. \quad (10.6)$$

Diese Gleichung impliziert, ebenso wie (10.1) auf Seite 260, dass zur Berechnung $\hat{V}_0(s)$ das im Die mittlere Belohnung wird zur Belohnung aller Nachfolgestaaten addiert, abgezinst von der Faktor $\tilde{\gamma}$. Wenn $V^{\tilde{v}}(\tilde{y}(s, a))$ bekannt ist, dann $V^{\tilde{v}}(s)$ ergibt sich eindeutig aus einfachen lokalen Operationen Timing über alle möglichen Aktionen a im Zustand s . Dies entspricht dem Bellman Prinzip, weshalb (10.6) auch Bellman-Gleichung genannt wird.

Die optimale Richtlinie $\tilde{v}^{\tilde{v}}(s)$ führt eine Aktion im Zustand s aus, die zur Folge hat Maximalwert $V^{\tilde{v}}$. Daher,

$$\tilde{v}^{\tilde{v}}(s) = \operatorname{argmax}_A [r(s, a) + \tilde{\gamma} V^{\tilde{v}}(\tilde{y}(s, a))]. \quad (10.7)$$

Aus der Rekursionsgleichung (10.6) eine Iterationsregel zur Approximation von $V^{\tilde{v}}$: folgt auf einfache Weise:

$$\hat{V}(s) = \max_A [r(s, a) + \tilde{\gamma} \hat{V}(\tilde{y}(s, a))]. \quad (10.8)$$

Zu Beginn werden beispielsweise die Näherungswerte $\hat{V}(s)$ für alle Zustände initialisiert mit dem Wert Null. Nun wird $\hat{V}(s)$ für jeden Zustand wiederholt rekursiv aktualisiert Zurückgreifen auf den Wert $\hat{V}(\tilde{y}(s, a))$ des besten Nachfolgezustandes. Dieser Prozess von Berechnung von $\hat{V}$ wird *Werteiteration* genannt und ist in Abb. 10.7 schematisch dargestellt

Abb. 10.7 Der Algorithmus für Wertiteration

BEWERTUNG()

Für alle $s \in S$

$V(s) = 0$

Wiederholen

Für alle $s \in S$

$\hat{V}(s) = \max_a [r(s, a) + \gamma V(\tilde{s}(s, a))]$

Bis sich $V(s)$ nicht ändert

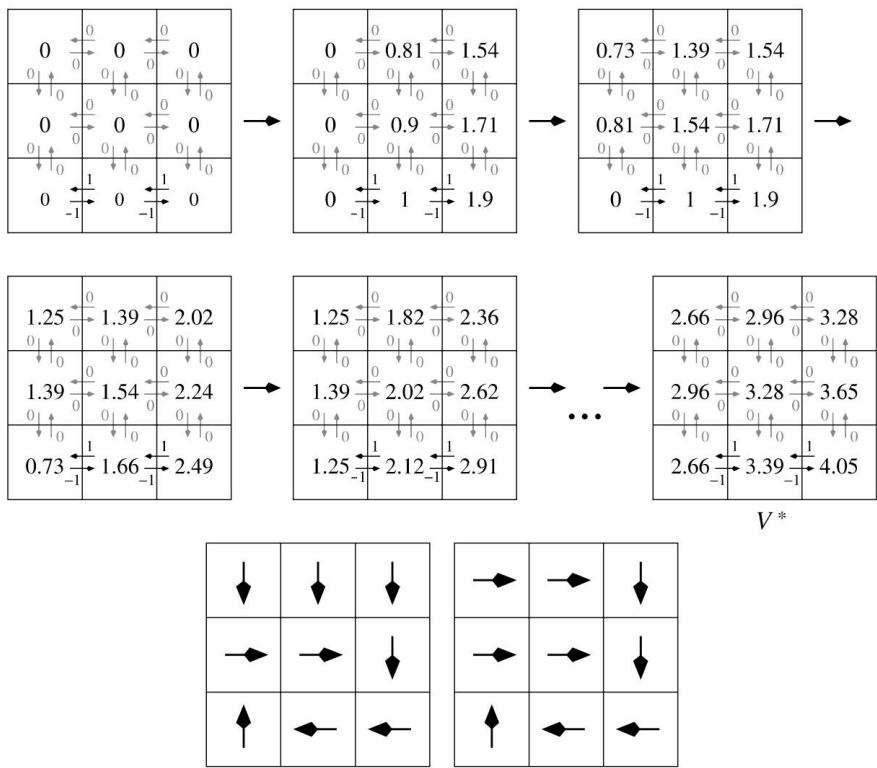


Abb. 10.8 Werteiteration im Beispiel mit 3 × 3 Zuständen. Die letzten beiden Bilder zeigen zwei optimale Richtlinien. Die Zahlen neben den Pfeilen geben die unmittelbare Belohnung $r(s, a)$ jeder Aktion an

auf Seite 264. Es kann gezeigt werden, dass die Werteiteration gegen V konvergiert [SB98]. Eine hervorragende Analyse dynamischer Programmieralgorithmen finden Sie in [Sze10], wobei basierend auf den Kontraktionseigenschaften der jeweiligen Algorithmen (z. B. Werteiteration) die Konvergenz mithilfe der Fixpunkttheorie von Banach nachgewiesen werden kann.

In Abb. 10.8 wird dieser Algorithmus auf Beispiel 10.1 auf Seite 257 mit $\tilde{\gamma} = 0,9$ angewendet. In jeder Iteration werden die Zustände zeilenweise von links unten nach rechts oben abgearbeitet. Dargestellt sind mehrere Anfangsiterationen und im zweiten Bild in der unteren Reihe die stabilen Grenzwerte für V .

In dieser Reihenfolge sehen wir deutlich den Fortschritt des Lernens. Der Agent durchsucht wiederholt alle Zustände, führt für jeden Zustand eine Werteiteration durch und speichert die in Form einer tabellierten Funktion V . Richtlinie, die dann weiter in eine Richtlinie kompiliert werden kann. effizient nutzbare Tabelle $\tilde{\gamma}$.

Um übrigens eine optimale Policy von V zu finden. Es wäre falsch, die Aktion im Zustand s zu wählen, die zu dem Zustand mit dem maximalen V gemäß (10.7) auf Wert. Korrekte Seite 263 führt. Die unmittelbare Belohnung $r(s, a)$ muss ebenfalls hinzugefügt werden, da wir in Abb. nach V suchen. 10.8 auf Seite (st) und nicht $V(s+1)$. Angewendet auf den Zustand $s = (2, 3)$ 264, das heißt

$$\begin{aligned} \tilde{\gamma}(2, 3) &= \operatorname{argmax}_{\text{links, rechts, oben}} [r(s, a) + \tilde{\gamma} V a \tilde{\gamma}(\text{links, rechts, oben})] \\ &= \operatorname{argmax}_{\text{links, rechts, oben}} \{1 + 0,9 \cdot 2,66, \tilde{\gamma}1 + 0,9 \cdot 4,05, 0 + 0,9 \cdot 3,28\} \\ &= \operatorname{argmax}_{\{\text{links, rechts, oben}\}} \{3,39, 2,65, 2,95\} = \text{links.} \end{aligned}$$

In (10.7) auf Seite 263 sehen wir, dass der Agent im Zustand s die unmittelbare Belohnung r und den Nachfolgezustand $s+1 = \tilde{\gamma}(s, a)$ kennen muss, um die optimale Aktion bei auszuwählen. Es muss auch ein Modell der Funktionen r und $\tilde{\gamma}$ vorhanden sein. Da dies für viele praktische Anwendungen nicht der Fall ist, werden Algorithmen benötigt, die auch ohne Kenntnis von r und $\tilde{\gamma}$ funktionieren können. Abschnitt 10.6 ist einem solchen Algorithmus gewidmet.

10.5 Ein lernender Laufroboter und seine Simulation

Eine grafische Benutzeroberfläche für einfache Experimente mit Reinforcement Learning ist in Abb. 10.9 auf Seite 266 dargestellt [TEF09]. Der Benutzer kann verstärkendes Lernen für unterschiedlich große zweidimensionale Zustandsräume beobachten. Zur besseren Verallgemeinerung werden Backpropagation-Netzwerke zur Zustandsspeicherung eingesetzt (siehe Abschn. 10.8). Besonders interessant für Experimente ist der unten rechts dargestellte Feedback-Editor, mit dem der Nutzer manuell Rückmeldungen über die Umgebung abgeben kann. Nicht gezeigt ist das Menü zum Einrichten der Parameter für die Werteiteration und das Backpropagation-Lernen.

Neben der Simulation wurden zwei kleine, echte Kriechroboter mit demselben zweidimensionalen diskreten Zustandsraum speziell für den Unterricht entwickelt [TEF09]. Die beiden Roboter sind in Abb. 10.10 auf Seite 266 dargestellt. Jeder bewegt sich mit einem Servoantrieb

2Weitere Informationen und verwandte Quellen zu Crawling-Robotern finden Sie unter www.hs-weingarten.de/~ertel/kibuch.

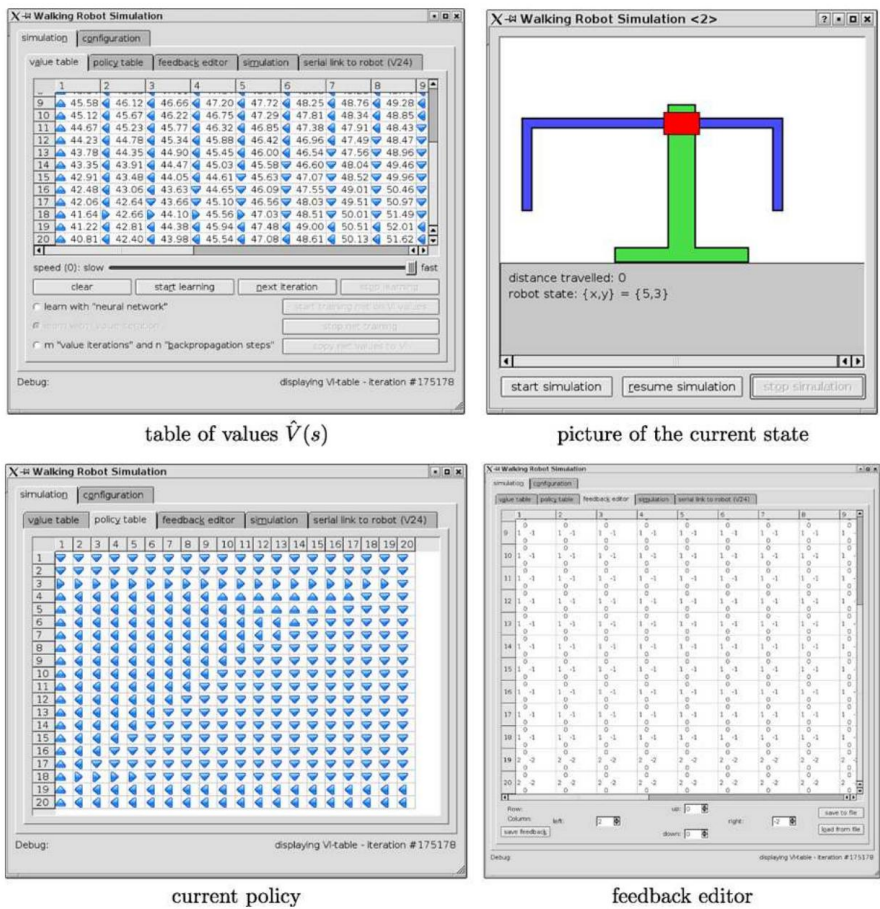


Abb. 10.9 Vier verschiedene Fenster des Laufrobotersimulators



Abb. 10.10 Zwei Versionen des Krabbleroboters

ator. Die Servos werden über einen Mikrocontroller oder über eine drahtlose Schnittstelle direkt von einem PC aus gesteuert. Mithilfe einer Simulationssoftware kann die Feedback-Matrix des Roboters auf dem PC visualisiert werden. Mit diesem gespeicherten Feedback kann eine Richtlinie auf dem PC trainiert werden (der schneller rechnet), dann wieder in den Roboter geladen und ausgeführt werden. Der Roboter kann jedoch auch autonom lernen. Bei einem Zustandsraum der Größe 5×5 dauert dies etwa 30 Sekunden.

Es ist interessant, den Unterschied zwischen der Simulation und dem „echten“ Roboter zu beobachten. Im Gegensatz zur Simulation erlernt der Crawler Verhaltensweisen, bei denen er seinen Arm nie vom Boden hebt, sich aber dennoch sehr effizient vorwärtsbewegt. Der Grund dafür liegt darin, dass die Spitze des „Unterarms“ je nach Untergrundoberfläche bei der Rückwärtsbewegung den Boden festhalten kann, bei der Vorwärtsbewegung jedoch durchrutscht. Dieser Effekt wird durch die Abstandsmesssensoren sehr gut wahrgenommen und beim Lernen entsprechend ausgewertet.

Die Adaptivität des Roboters führt zu überraschenden Effekten. Wir können beispielsweise beobachten, wie die Raupe trotz eines defekten Servos, das in einem bestimmten Winkel durchrutscht, dennoch das Laufen lernt (eher humpelt). Es ist sogar in der Lage, sich durch geänderte Richtlinien an veränderte Situationen anzupassen. Ein durchaus wünschenswerter Effekt ist die Fähigkeit, bei unterschiedlich glatten Oberflächen (z. B. unterschiedlich rauen Teppichen) jeweils eine optimale Vorgehensweise zu erlernen. Es zeigt sich auch, dass der reale Roboter angesichts eines kleinen Zustandsraums der Größe 5×5 tatsächlich sehr anpassungsfähig ist.

Der Leser kann (in Ermangelung eines echten Roboters) verschiedene Oberflächen oder Servodefekte modellieren, indem er die Rückkopplungswerte variiert und dann die daraus resultierenden Richtlinien beachtet (Übung 10.3 auf Seite 276).

10.6 Q-Learning

Eine Politik, die auf der Bewertung möglicher Nachfolgezustände basiert, ist eindeutig nicht anwendbar, wenn der Agent kein Modell der Welt hat, das heißt, wenn er nicht weiß, zu welchem Zustand eine mögliche Aktion führt. In den meisten realistischen Anwendungen kann der Agent nicht auf ein solches Weltmodell zurückgreifen. Beispielsweise kann ein Roboter, der komplexe Objekte greifen soll, nicht vorhersagen, ob das Objekt nach einem Greifvorgang sicher im Griff gehalten wird oder ob es an Ort und Stelle bleibt.

Liegt kein Modell der Welt vor, ist eine Bewertung einer im Zustand s_t durchgeführten Aktion erforderlich, auch wenn noch unbekannt ist, wohin diese Aktion führt. Daher arbeiten wir nun mit einer Bewertungsfunktion $Q(s_t, a_t)$ für Zustände mit ihren zugehörigen Aktionen. Bei dieser Funktion erfolgt die Wahl der optimalen Aktion durch die Regel

$$\hat{y}(s) = \underset{A}{\operatorname{argmax}} Q(s, a). \quad (10.9)$$

Um die Bewertungsfunktion zu definieren, verwenden wir wiederum eine schrittweise Diskontierung der Bewertung für Zustands-Aktionspaare, die weiter in der Zukunft liegen, genau wie in (10.1) auf Seite 260. Wir wollen also $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$ maximieren.

Um die Aktion im Zustand s_t zu bewerten, definieren wir daher in Analogie zu (10.4) on

Seite 263:

$$Q(s, a) = \max_{a \in A} \{ r(s, a) + \gamma \sum_{s'} P(s' | s, a) Q(s', a) \} \quad (10.10)$$

Analog zum Ansatz zur Wertiteration bringen wir diese Gleichung durch in eine einfache rekursive Form

$$\begin{aligned} Q(s, a) &= \max_{a \in A} \{ r(s, a) + \gamma \sum_{s'} P(s' | s, a) Q(s', a) \} \\ &= r(s, a) + \gamma \max_{a \in A} \{ \sum_{s'} P(s' | s, a) Q(s', a) \} \\ &= r(s, a) + \gamma \max_{a \in A} \{ \sum_{s'} P(s' | s, a) [r(s', a') + \gamma \sum_{s''} P(s'' | s', a') Q(s'', a')] \} \\ &= r(s, a) + \gamma \max_{a \in A} \{ \sum_{s'} P(s' | s, a) Q(s', a) \} \\ &= r(s, a) + \gamma \max_{a \in A} \{ \sum_{s'} P(s' | s, a) Q(s', a) \} \\ &= r(s, a) + \gamma \max_{a \in A} \{ \sum_{s'} P(s' | s, a) Q(s', a) \} \end{aligned} \quad (10.11)$$

Was ist dann der Vorteil gegenüber der Wertiteration? Die alte Gleichung wird nur leicht umgeschrieben, aber das erweist sich als genau der richtige Ansatz für einen neuen Algorithmus. Anstatt V zu speichern, wird nun die Funktion Q gespeichert, und der Agent kann seine Aktionen aus den Funktionen γ und r auswählen, ohne ein Modell der Welt zu haben. Allerdings gibt es noch keinen Prozess, der Q direkt, also ohne Kenntnis von V , lernen kann. Aus der

rekursiven Formulierung von $Q(s, a)$ lässt sich ein Iterationsalgorithmus zur Bestimmung von $Q(s, a)$ in einer ableitbaren unkomplizierten Art und Weise. Wir initialisieren eine Tabelle $Q(s, a)$ für alle Zustände beliebig, beispielsweise mit Nullen, und führen sie iterativ aus

$$Q(s, a) = r(s, a) + \gamma \max_{a' \in A} Q(s', a') \quad (10.12)$$

Es bleibt zu beachten, dass wir die Funktionen r und γ nicht kennen. Wir lösen dieses Problem ganz pragmatisch, indem wir den Agenten in seiner Umgebung im Zustand s die Aktion a ausführen lassen. Der Nachfolgezustand ist dann eindeutig $\gamma(s, a)$ und der Agent erhält seine Belohnung von der Umgebung. Der in Abb. 10.11 auf Seite 269 dargestellte Algorithmus implementiert diesen Algorithmus für Q-Learning.

Die Anwendung des Algorithmus auf Beispiel 10.1 auf Seite 257 mit $\gamma = 0,9$ und $n_x = 3$, $n_y = 2$ (also in einem 2×3 -Raster) ist in Abb. 10.12 auf Seite 269 beispielhaft dargestellt. Im ersten Bild werden alle Q -Werte auf Null initialisiert. Im zweiten Bild werden nach der ersten Aktionssequenz die vier r -Werte, die ungleich Null sind, als Q -Werte sichtbar. Im letzten Bild ist die erlernte optimale Richtlinie dargestellt. Der folgende Satz, dessen Beweis in [Mit97] zu finden ist, zeigt, dass dieser Algorithmus nicht nur im Beispiel, sondern allgemein konvergiert.

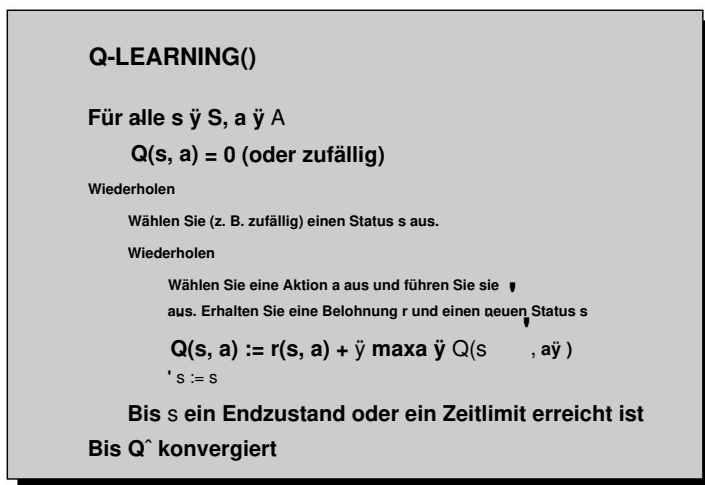
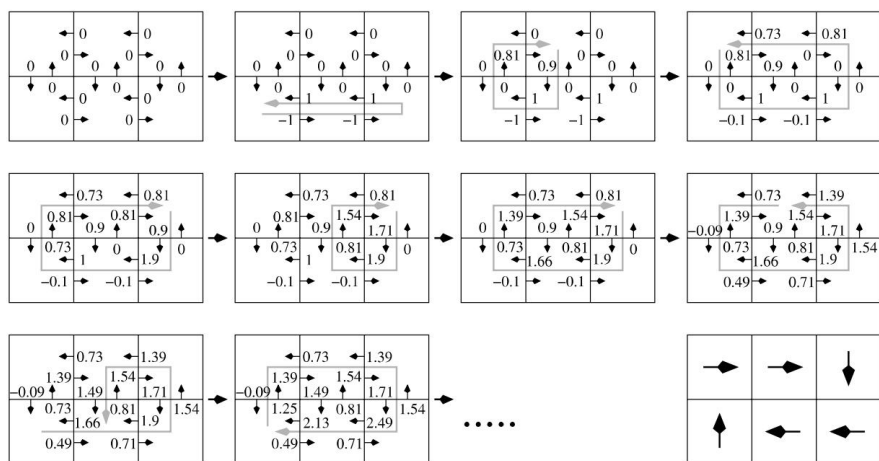


Abb. 10.11 Der Algorithmus für Q-Learning

Abb. 10.12 Q-Learning angewendet auf das Beispiel mit $n_x = 3, n_y = 2$. Die *grauen Pfeile* markieren die durchgeführten Aktionen in jedem Bild. Die aktualisierten Q- Werte werden angezeigt. Im *letzten Bild* ist die aktuelle, ebenfalls optimale Politik dargestellt

Satz 10.1 Es sei ein deterministischer MDP mit begrenzter unmittelbarer Belohnung $r(s, a)$ gegeben. Zum Lernen wird Gleichung (10.12) auf Seite 268 mit $0 < \gamma < 1$ verwendet. Sei $Q^n(s, a)$ der Wert für $Q(s, a)$ nach n Aktualisierungen. Wenn jedes Zustandsaktionspaar unendlich oft besucht wird, dann konvergiert $Q^n(s, a)$ für alle Werte s und a für $n \rightarrow \infty$ gegen $Q(s, a)$.

Beweis Da jeder Zustands-Aktions-Übergang unendlich oft auftritt, betrachten wir aufeinanderfolgende Zeitintervalle mit der Eigenschaft, dass in jedem Intervall alle Zustands-Aktions-Übergänge mindestens einmal auftreten. Wir zeigen nun, dass der maximale Fehler für alle Einträge in der Q^* -Tabelle in jedem dieser Intervalle mindestens um den Faktor γ reduziert wird. Lassen

$$e_n = \max_{s,a} |Q^n(s, a) - Q^*(s, a)|$$

sei der maximale Fehler in der Tabelle Q^n und $s' = \gamma(s, a)$. Für jeden Tabelleneintrag $Q^n(s, a)$ Wir berechnen seinen Beitrag zum Fehler nach einem Intervall als

$$\begin{aligned} |Q^{n+1}(s, a) - Q^*(s, a)| &= r + \gamma \max_{a'} Q^n(s\gamma, a\gamma) - Q^*(s, a) + \gamma \max_{a'} Q(s\gamma, a\gamma) - Q^*(s\gamma, a\gamma) \\ &= \gamma \max_{a'} |Q^n(s\gamma, a\gamma) - Q^*(s\gamma, a\gamma)| \\ &\leq \gamma \max_{s', a'} |Q^n(s', a') - Q^*(s', a')| \\ &= \gamma e_n \end{aligned}$$

Die erste Ungleichung ist wahr, weil für beliebige Funktionen f und g

$$\max_x |f(x) - g(x)| \geq \max_x |f(x) - g(x)|$$

und die zweite Ungleichung ist wahr, weil durch zusätzliche Variation des Zustands s das resultierende Maximum nicht kleiner werden kann. Somit wurde gezeigt, dass $n+1$ Da der Fehler γe_n in jedem Intervall mindestens um den Faktor γ reduziert wird, beträgt er nach k Intervallen höchstens $\gamma^k e_0$ und ist daher auf 0 beschränkt. Da jeder Staat unendlich oft besucht wird, gibt es unendlich viele Intervalle und null. n konvergiert

Gemäß Satz 10.1 auf Seite 269 konvergiert Q-Learning unabhängig von den beim Lernen gewählten Aktionen. Das bedeutet, dass es für die Konvergenz keine Rolle spielt, welche Aktionen der Agent auswählt, solange jede davon unendlich oft ausgeführt wird. Die Konvergenzgeschwindigkeit hängt jedoch sicherlich davon ab, welche Wege der Agent beim Lernen einschlägt (siehe Abschn. 10.7).

10.6.1 Q-Learning in einer nichtdeterministischen Umgebung

In vielen Robotikanwendungen ist die Umgebung des Agenten nichtdeterministisch. Dies bedeutet, dass die Reaktion der Umwelt auf die Aktion a in Zuständen zu zwei unterschiedlichen Zeitpunkten zu unterschiedlichen Folgezuständen und Belohnungen führen kann. Ein solcher nichtdeterministischer Markov-Prozess wird durch eine probabilistische Übergangsfunktion $\gamma(s, a)$ und eine probabilistische unmittelbare Belohnung $r(s, a)$ modelliert. Um die Q -Funktion zu definieren, muss jeweils der Erwartungswert über alle möglichen Folgezustände berechnet werden. Gleichung (10.

auf Seite 268 wird somit verallgemeinert

$$Q(s, a) = E(r(s, a) + \gamma \max_{a'} P(s' | s, a) Q(s', a')) \quad (10.13)$$

Dabei ist $P(s' | s, a)$ die Wahrscheinlichkeit, mit der Aktion a vom Zustand s zum Nachfolgezustand s' zu wechseln. Leider gibt es für Q-Learning im nichtdeterministischen Fall keine Konvergenzgarantie, wenn wir wie zuvor gemäß (10.12) auf Seite 268 vorgehen. Dies liegt daran, dass bei aufeinanderfolgenden Durchläufen der äußeren Schleife des Algorithmus in Abb. 10.11 auf Seite 269 Belohnungs- und Nachfolgezustand für denselben Zustand s und dieselbe Aktion a völlig unterschiedlich sein können. Dabei kann es zu einer alternierenden Sequenz kommen, die zwischen mehreren Werten hin- und herspringt. Um solche stark springenden Q -Werte zu vermeiden, fügen wir den alten gewichteten Q -Wert auf der rechten Seite von (10.12) auf Seite 268 hinzu. Dies stabilisiert die Iteration. Die Lernregel lautet dann

$$Q^n(s, a) = (1 - \gamma_n) Q^{n-1}(s, a) + \gamma_n r(s, a) + \gamma \max_{a'} Q^{n-1}(\tilde{s}(s, a), a') \quad (10.14)$$

mit einem zeitlich veränderlichen Gewichtungsfaktor

$$\gamma_n = \frac{1}{1 + \text{Mrd.}(s, a)}$$

Der Wert $\text{Mrd.}(s, a)$ gibt an, wie oft die Aktion a im Zustand s bei der n -ten Iteration ausgeführt wurde. Für kleine Werte von $\text{Mrd.}(s, a)$ (also zu Beginn des Lernens) kommt der stabilisierende Term $Q^{n-1}(s, a)$ nicht ins Spiel, da wir wollen, dass der Lernprozess schnell voranschreitet. Später wird $\text{Mrd.}(s, a)$ jedoch größer und verhindert dadurch zu große Sprünge in der Folge der Q -Werte. Bei der Integration von (10.14) in Q-Learning müssen für alle Zustands-Aktionspaare die Werte $\text{Mrd.}(s, a)$ gespeichert werden. Dies kann durch eine Erweiterung der Tabelle der Q -Werte erreicht werden.

Zum besseren Verständnis von (10.14) vereinfachen wir dies, indem wir annehmen, dass $\gamma_n = \gamma$ ist konstant und transformiere es wie folgt:

$$\begin{aligned} Q^n(s, a) &= (1 - \gamma) Q^{n-1}(s, a) + \gamma r(s, a) + \gamma \max_{a'} Q^{n-1}(\tilde{s}(s, a), a') \\ &= Q^{n-1}(s, a) + \gamma r(s, a) + \gamma \max_{a'} Q^{n-1}(\tilde{s}(s, a), a') - Q^{n-1}(s, a) \end{aligned}$$

TD-Fehler

Der neue Q -Wert $Q^n(s, a)$ kann eindeutig als der alte $Q^{n-1}(s, a)$ plus γ mal einem Korrekturterm dargestellt werden, der der Änderung des Q -Werts in diesem Schritt entspricht. Der Korrekturterm wird TD-Fehler oder zeitlicher Differenzfehler genannt, und die obige Gleichung zur Änderung des Q -Werts ist ein Sonderfall von TD-Learning, einer wichtigen Klasse von Lernalgorithmen [SB98]. Für $\gamma = 1$ erhalten wir das oben beschriebene Q-Learning. Für $\gamma = 0$ bleiben die Q -Werte völlig unverändert. Somit findet kein Lernen statt.

10.7 Exploration und Ausbeutung

Für Q-Learning wurde bisher nur ein grobes Algorithmuschema angegeben. Besonders Es fehlt eine Beschreibung der jeweiligen Wahl des Startzustands und der auszuführenden Aktionen in der inneren Schleife von Abb. 10.11 auf Seite 269. Für die Auswahl der nächsten Aktion gibt es zwei Möglichkeiten. Zu den möglichen Aktionen zählen man kann zufällig ausgewählt werden. Langfristig führt dies zu einer einheitlichen Erkundung aller möglichen Aktionen oder Richtlinien, jedoch mit sehr langsamer Konvergenz. Eine Alternative Dazu gehört die Nutzung zuvor erlernter Q^* -Werte. Hier immer der Agent wählt die Aktion mit dem höchsten Q^* -Wert. Dies führt zu einer relativ schnellen Konvergenz einer bestimmten Flugbahn. Andere Wege bleiben jedoch durchgehend unbegangen bis zum Ende. Im Extremfall können wir dann nicht optimale Richtlinien erhalten. In Theo rem 10.1 auf Seite 269 ist es daher erforderlich, dass jedes Zustands-Aktionspaar besucht wird unendlich oft. Es wird empfohlen, eine Kombination aus Erkundung und zu verwenden Die Ausbeutung mit einem hohen Explorationsanteil am Anfang steigern und diesen immer weiter reduzieren mehr im Laufe der Zeit.

Auch die Wahl des Ausgangszustandes beeinflusst die Lerngeschwindigkeit. In der ersten In den drei Bildern in Abb. 10.12 auf Seite 269 ist deutlich zu erkennen, dass für die ersten Iterationen nur die Q-Werte in der unmittelbaren Umgebung von Zustands-Aktionspaaren geändert werden durch sofortige Belohnung. Wenn man weiter von einem solchen Punkt entfernt anfängt, ergibt sich viel unnötige Arbeit. Dies legt nahe, Vorwissen über staatliches Handeln zu übertragen Paare mit sofortiger Belohnung in Startzustände in der Nähe dieser Punkte. Im Kurs des Lernens können weiter entfernte Ausgangszustände gewählt werden.

10.8 Approximation, Generalisierung und Konvergenz

Da Q-Learning bisher beschrieben wurde, ist explizit eine Tabelle mit allen Q-Werten erforderlich in einer Tabelle gespeichert. Dies ist nur möglich, wenn mit einem endlichen Zustandsraum gearbeitet wird endlich viele Aktionen. Wenn der Zustandsraum jedoch beispielsweise im Fall unendlich ist von kontinuierlichen Variablen ist es weder möglich, alle Q-Werte zu speichern noch zu besuchen alle Zustands-Aktionspaare während des Lernens.

Dennoch gibt es eine einfache Möglichkeit, Q-Learning und Werteiteration auf kontinuierliche Variablen anzuwenden. Die $Q(s, a)$ -Tabelle wird durch ein neuronales Netzwerk ersetzt, zum Beispiel a Backpropagation-Netzwerk mit den Eingabevariablen s, a und dem Q-Wert als Ziel Ausgabe. Für jede Aktualisierung eines Q-Wertes wird dem neuronalen Netzwerk ein Training vorgelegt Beispiel mit (s, a) als Eingabe und $Q(s, a)$ als Zielausgabe. Am Ende haben wir eine endliche Darstellung der Funktion $Q(s, a)$. Da wir immer nur endlich viele haben Trainingsbeispiele, aber die Funktion $Q(s, a)$ ist für unendlich viele Eingaben definiert, wir Damit erhält man bei geeigneter Wahl der Netzwerkgröße automatisch eine Verallgemeinerung (siehe Kap. 9). Anstelle eines neuronalen Netzwerks können wir auch ein anderes überwachtes Netzwerk verwenden Lernalgorithmus oder ein Funktionsnäherungsgerät wie eine Support-Vektor-Maschine oder ein Gaußscher Prozess.

Allerdings kann der Schritt von endlich vielen Trainingsbeispielen zu einer kontinuierlichen Funktion in bestimmten Situationen sehr teuer werden. Q-Learning mit Funktionsnäherung konvergiert möglicherweise nicht, da Satz 10.1 auf Seite 269 nur dann wahr ist, wenn jedes Zustands-Aktionspaar unendlich oft besucht wird.

Allerdings kann es auch bei endlich vielen Zustands-Aktionspaaren beim Einsatz von Q-Learning auf einem POMDP zu Konvergenzproblemen kommen. Q-Learning kann – in beiden beschriebenen Varianten – auf deterministische und nichtdeterministische Markov-Prozesse (MDPs) angewendet werden. Bei einem POMDP kann es vorkommen, dass der Agent, beispielsweise aufgrund verrauschter Sensoren, viele verschiedene Zustände als einen einzigen wahrnimmt. Oft werden viele Zustände in der realen Welt gezielt auf eine sogenannte *Beobachtung abgebildet*. Der resultierende Beobachtungsraum ist dann deutlich kleiner als der Zustandsraum, wodurch das Lernen schneller wird und eine Überanpassung vermieden werden kann (siehe [SB98]).

Durch die Bündelung mehrerer Zustände kann der Agent jedoch nicht mehr zwischen den tatsächlichen Zuständen unterscheiden, und eine Aktion kann ihn in viele verschiedene Nachfolgezustände führen, je nachdem, in welchem Zustand er sich tatsächlich befindet. Dies kann zu Konvergenzproblemen für den Wert führen Iteration oder für Q-Learning. In der Literatur (z. B. in [SB98]) werden viele unterschiedliche Lösungsansätze vorgeschlagen.

Sehr vielversprechend sind auch sogenannte Policy-Verbesserungsmethoden und daraus abgeleitete Policy-Gradienten-Methoden, bei denen Q-Werte nicht verändert werden, sondern die Policy direkt verändert wird. Bei diesem Schema wird im Raum aller Policies nach einer Policy gesucht, die die kumulative abgezinste Belohnung maximiert ((10.1) auf Seite 260). Eine Möglichkeit, dies zu erreichen, besteht darin, dem Gradienten der kumulativen Belohnung bis zum Maximum zu folgen. Die so gefundene Policy optimiert dann deutlich die kumulierte Belohnung. In [PS08] wird gezeigt, dass dieser Algorithmus das Lernen in Anwendungen mit großen Zustandsräumen, wie sie beispielsweise bei humanoiden Robotern auftreten, erheblich beschleunigen kann.

10.9 Anwendungen

Der praktische Nutzen von Reinforcement Learning wurde mittlerweile vielfach nachgewiesen. Aus einer Vielzahl von Beispielen hierzu stellen wir Ihnen kurz eine kleine Auswahl vor.

TD-Learning wurde zusammen mit einem Backpropagation-Netzwerk mit 40 bis 80 versteckten Neuronen sehr erfolgreich in TD-Gammon, einem Backgammon-Spielprogramm, eingesetzt [Tes95]. Die einzige unmittelbare Belohnung für das Programm ist das Ergebnis am Ende des Spiels. Eine optimierte Version des Programms mit einem Lookahead mit zwei Zügen wurde in 1,5 Millionen Spielen gegen sich selbst trainiert. Anschließend wurden Weltklassespieler und -spieler sowie die drei besten menschlichen Spieler besiegt.

Es gibt viele Anwendungen in der Robotik. In der RoboCup Soccer Simulation League beispielsweise setzen die besten Roboterfußballteams mittlerweile erfolgreich Reinforcement Learning ein [SSK05, Robb]. Das Balancieren einer Stange, das für einen Menschen relativ einfach ist, wurde mit Verstärkungslernen schon oft erfolgreich gelöst.

Einen eindrucksvollen Beweis für die Lernfähigkeit von Robotern lieferte Russ Tedrake auf der IROS 2008 mit seinem Vortrag über ein Modellflugzeug, das lernt, an einem genauen Punkt zu landen, genau wie ein Vogel, der auf einem Ast landet [Ted08]. Weil Luft

Da die Strömungen bei solch einem hochdynamischen Landeanflug sehr turbulent werden, ist die zugehörige Differentialgleichung, die Navier-Stokes-Gleichung, unlösbar. Die Landung kann daher nicht auf klassische mathematische Weise gesteuert werden. Tedrakes Kommentar dazu:

„Vögel lösen Navier-Stokes nicht!“

Vögel können offensichtlich auch ohne die Navier-Stokes-Gleichung fliegen und landen lernen. Tedrake zeigte, dass dies nun auch für Flugzeuge möglich ist.

Heute ist es auch möglich, mithilfe von Q-Learning und Funktionsnäherung [RMD07] zu lernen, ein echtes Auto in nur 20 Minuten zu steuern. Dieses Beispiel zeigt, dass reale Industrieanwendungen, bei denen wenige Messungen in Aktionen abgebildet werden müssen, sehr gut in kurzer Zeit erlernt werden können.

Echte Roboter haben immer noch Schwierigkeiten beim Lernen in hochdimensionalen Zustands- und Aktionsräumen, da echte Roboter im Vergleich zu einer Simulation relativ langsam Feedback von der Umgebung erhalten. Aus zeitlichen Gründen sind die vielen Millionen notwendigen Trainingszyklen daher nicht realisierbar. Hier sind neben schnellen Lernalgorithmen auch Methoden gefragt, die es ermöglichen, zumindest Teile des Lernens offline, also ohne Rückmeldung aus der Umgebung, ablaufen zu lassen.

10.10 Fluch der Dimensionalität

Trotz der Erfolge in den letzten Jahren bleibt Reinforcement Learning ein aktives Forschungsgebiet in der KI, nicht zuletzt, weil selbst die besten heute bekannten Lernalgorithmen aufgrund ihrer gigantischen Rechenzeit noch immer für hochdimensionale Zustands- und Aktionsräume unpraktisch sind. Dieses Problem ist als „Fluch der Dimensionalität“ bekannt.

Auf der Suche nach Lösungen für dieses Problem beobachten Wissenschaftler Tiere und Menschen beim Lernen. Dabei fällt uns auf, dass Lernen in der Natur auf vielen Abstraktionsebenen stattfindet. Auf der untersten Stufe erlernt ein Baby zunächst einfache motorische und sprachliche Fähigkeiten. Wenn diese gut erlernt sind, werden sie gespeichert und können später jederzeit abgerufen und verwendet werden. In die Sprache der Informatik übersetzt heißt das, dass jede erlernte Fähigkeit in einem Modul gekapselt wird und dann auf einer höheren Ebene eine Handlung darstellt. Durch die Verwendung solch komplexer Aktionen auf einer höheren Ebene wird der Aktionsraum stark reduziert und somit das Lernen beschleunigt. Auf ähnliche Weise können Zustände abstrahiert und damit der Zustandsraum verkleinert werden. Dieses Lernen auf mehreren Ebenen wird hierarchisches Lernen genannt [BM03].

Ein weiterer Ansatz zur Modularisierung des Lernens ist verteiltes Lernen oder Multi-Agent-Lernen [PL05]. Beim Erlernen der motorischen Fähigkeiten eines humanoiden Roboters müssen bis zu 50 verschiedene Motoren gleichzeitig gesteuert werden, wodurch ein 50-dimensionaler Zustandsraum und auch ein 50-dimensionaler Aktionsraum entstehen. Um diese gigantische Komplexität zu reduzieren, wird die zentrale Steuerung durch eine verteilte Steuerung ersetzt. Beispielsweise könnte jeder einzelne Motor eine eigene Steuerung erhalten, die ihn direkt steuert, möglichst unabhängig von den anderen Motoren. In der Natur finden wir diese Art der Bekämpfung bei Insekten. Beispielsweise werden die vielen Beine eines Tausendfüßlers nicht von einem zentralen Gehirn gesteuert, sondern jedes Beinpaar hat sein eigenes Kl

Ähnlich wie bei der uninformatierten kombinatorischen Suche besteht beim Reinforcement Learning die Aufgabe, aus einer Vielzahl von Richtlinien die beste zu finden. Die Lernaufgabe wird erheblich einfacher, wenn der Agent vor Beginn des Lernens eine mehr oder weniger gute Richtlinie hat. Dann können die hochdimensionalen Lernaufgaben schneller gelöst werden. Aber wie finden wir eine solche anfängliche Politik? Hier gibt es zwei Hauptmöglichkeiten.

Die erste Möglichkeit ist die klassische Programmierung. Der Programmierer stellt dem Agenten eine Richtlinie zur Verfügung, die ein Programm umfasst, das er für gut hält. Dann erfolgt eine Umstellung, beispielsweise auf Q-Learning. Der Agent wählt, zumindest zu Beginn des Lernens, seine Handlungen entsprechend der programmierten Politik und wird so in „interessante“ Bereiche des staatlichen Handlungsraums geführt. Dies kann zu einer dramatischen Verkleinerung des Suchraums des Reinforcement Learning führen.

Wenn die herkömmliche Programmierung zu komplex wird, können wir damit beginnen, den Roboter oder Agenten zu trainieren, indem wir ihm die richtigen Aktionen von einem Menschen vorschreiben lassen. Im einfachsten Fall erfolgt dies durch manuelle Fernbedienung des Roboters. Der Roboter speichert dann die vorgeschriebene Aktion für jeden Zustand und verallgemeinert sie mithilfe eines überwachten Lernalgorithmus wie Backpropagation oder Entscheidungsbaulernen. Dieses sogenannte Demonstrationlernen [BCDS08, SE10] bietet somit auch eine erste Richtlinie für das anschließende Verstärkungslernen.

10.11 Zusammenfassung und Ausblick

Heute stehen uns gut funktionierende und etablierte Lernalgorithmen für das Training unserer Maschinen zur Verfügung. Allerdings ist die Aufgabe für den menschlichen Trainer oder Entwickler bei komplexen Anwendungen immer noch anspruchsvoll. Es gibt nämlich viele Möglichkeiten, das Training eines Roboters zu gestalten, und ohne Experimente wird es nicht gelingen. Dieses Experimentieren kann in der Praxis sehr mühsam sein, da jedes neue Lernprojekt entworfen und programmiert werden muss. Hier sind Werkzeuge gefragt, die dem Trainer neben den verschiedenen Lernalgorithmen auch die Möglichkeit bieten, diese mit klassischem Programmieren und Demonstrationlernen zu kombinieren. Eines der ersten Tools dieser Art ist die Teaching-Box [ESCT09], die neben einer umfangreichen Programmbibliothek auch Vorlagen für die Konfiguration von Lernprojekten und für die Kommunikation zwischen Roboter und Umgebung bietet. Beispielsweise kann der menschliche Lehrer dem Roboter zusätzlich zum Feedback aus der Umgebung weiteres Feedback über die Tastatur oder über eine Sprachschnittstelle geben.

Reinforcement Learning ist ein faszinierendes und aktives Forschungsgebiet, das in Zukunft zunehmend genutzt werden wird. Immer mehr Robotersteuerungen, aber auch andere Programme, lernen durch Feedback aus der Umgebung. Heutzutage gibt es eine Vielzahl von Variationen der vorgestellten Algorithmen und auch völlig unterschiedliche Algorithmen. Das Skalierungsproblem bleibt ungelöst. Für kleine Aktions- und Zustandsräume mit wenigen Freiheitsgraden können beeindruckende Ergebnisse erzielt werden. Wenn die Anzahl der Freiheitsgrade im Zustandsraum beispielsweise bei einem einfachen humanoiden Roboter auf 18 anwächst, wird das Lernen sehr teuer.

Für weitere Grundlagenvorlesungen empfehlen wir die kompakte Einführung in das Reinforcement Learning in Tom Mitchells Buch [Mit97]. Das Standardwerk von Sutton und Barto [SB98] ist gründlich und umfassend, ebenso wie der Übersichtsartikel von Kaelbling, Littman und Moore [KLM96].

10.12 Übungen

Aufgabe 10.1 (a)

Berechnen Sie die Anzahl verschiedener Richtlinien für n Zustände und n Aktionen. Somit sind Übergänge von jedem Staat zum anderen möglich. (b) Wie

ändert sich die Anzahl der Richtlinien in Teilproblem (a), wenn leere Aktionen, also Aktionen von einem Zustand zu sich selbst, nicht erlaubt sind? (c) Geben

Sie mithilfe von Pfeildiagrammen wie denen in Abb. 10.3 auf Seite 259 alle Richtlinien für n zwei Staaten.

(d) Geben Sie mithilfe von Pfeildiagrammen alle Richtlinien ohne leere Aktionen für drei Zustände an.

Übung 10.2 Verwenden Sie die Wertiteration manuell in Beispiel 10.1 auf Seite 257 mit $n_x = n_y = 2$.

Übung 10.3 Führen Sie verschiedene Experimente mit einem Wertiterationssimulator durch.

(a) Installieren Sie den Wertiterationssimulator von [Tok06]. (b)

Reproduzieren Sie die Ergebnisse aus Übung 10.2, indem Sie zunächst das Feedback mit eingeben des Feedback-Editors und anschließende Durchführung der

Wertiteration. (c) Modellieren Sie Oberflächen unterschiedlicher Glätte und beobachten Sie, wie sich die Richtlinie ändert. (d) Erweitern Sie mit einer ähnlichen Rückkopplungsmatrix den

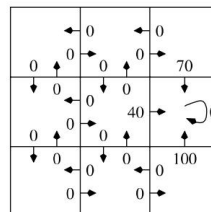
Zustandsraum schrittweise auf etwa 100×100 und passen Sie den Abzinsungsfaktor γ so an, dass sich eine sinnvolle Richtlinie ergibt.

Übung 10.4 Zeigen Sie, dass für die Beispielrechnung in Abb. 10.8 auf Seite 264 $\gamma = 4,05499$ ist. Der genaue Wert ist $\gamma = 1,9/(1 - 0,9)$

Übung 10.5

Führen Sie Q-Learning im 3×3 -Raster rechts durch.

Der Zustand in der Mitte rechts ist ein absorbierender Zielzustand.



Aufgabe 10.6 Gegeben ist ein Roboterarm mit n Gelenken (Dimensionen) und γ diskreten Zuständen pro Gelenk. Aktionen von jedem Zustand zu jedem Zustand sind möglich (wenn der Roboter nichts tut, wird dies als (leere) Aktion gewertet).

-
- (a) Geben Sie eine Formel für die Anzahl der Zustände und die Anzahl der Aktionen in jedem Zustand an und für die Anzahl der Richtlinien für den Roboter.
- (b) Erstellen Sie eine Tabelle mit der Anzahl der Strategien für $n = 1, 2, 3, 4, 8$ und $\bar{y} = 1, 2, 3, 4, 10$.
- (c) Um die Anzahl möglicher Strategien zu reduzieren, nehmen Sie an, dass die Anzahl möglicher Aktionen pro Gelenk immer gleich 2 ist und dass der Roboter jeweils nur ein Gelenk bewegen kann. Geben Sie eine neue Formel für die Anzahl der Strategien an und erstellen Sie die zugehörige Tabelle.
- (d) Begründen Sie mit dem berechneten Ergebnis, dass ein Agent, der autonom und adaptiv mit $n = 8$ und $l = 10$ agiert, durchaus als intelligent bezeichnet werden kann.

11.1 Einführung

Aufgabe 1.3 Viele bekannte Inferenzprozesse, Lernprozesse etc. sind NP-vollständig oder sogar unentscheidbar. Was bedeutet das für KI?

Aufgabe 1.4 Wenn ein Problem NP-vollständig ist oder als „schwer“ beschrieben werden kann, bedeutet das, dass es Fälle gibt, in denen das Problem nicht in akzeptabler Zeit gelöst werden kann. Dies ist der sogenannte *Worst Case*. Bei manchen Anwendungen müssen wir damit leben, dass im schlimmsten Fall keine effiziente Lösung möglich ist. Das bedeutet, dass es auch in Zukunft praktisch relevante Probleme geben wird, die in bestimmten Sonderfällen unlösbar sind.

KI wird daher weder eine universelle Formel finden, noch eine Supermaschine bauen, mit der alle Probleme lösbar werden. Es stellt sich vielmehr die Aufgabe, Systeme mit einer höheren Wahrscheinlichkeit, eine Lösung zu finden, bzw. mit einer höheren Wahrscheinlichkeit, schnelle, optimale Lösungen zu finden, aufzubauen. Wir Menschen kommen im Alltag recht gut mit suboptimalen Lösungen zurecht. Der Grund liegt ganz einfach in den zu hohen Kosten für die Suche nach der optimalen Lösung. Beispielsweise brauche ich nur sieben Minuten, um mit einer Karte in einer unbekannten Stadt den Weg von Punkt A nach Punkt B zu finden. Der kürzeste Weg hätte nur sechs Minuten gedauert. Den kürzesten Weg zu finden, hätte jedoch vielleicht eine Stunde gedauert. Der Beweis der Optimalität des Pfades dürfte noch aufwändiger sein.

Aufgabe 1.5

- (a) Die Ausgabe hängt nicht nur von der Eingabe ab, sondern auch vom Inhalt des Speichers. Für eine Eingabe x könnte die Ausgabe je nach Inhalt des Speichers y_1 oder y_2 sein. Es ist also nicht eindeutig und daher keine Funktion. (b) Betrachtet man den Inhalt des Speichers als weiteren Input, dann ist der Output eindeutig (da der Agent deterministisch ist) und der Agent stellt eine Funktion dar.

Aufgabe 1.6 (a)

Geschwindigkeit $v_x(t) = \frac{dx}{dt} \lim_{t \rightarrow 0} \frac{x(t) - x(t_0)}{t - t_0} = \frac{x(t) - x(t_0)}{t - t_0} \cdot \frac{1}{t - t_0}$. v_y wird berechnet

analog. $\ddot{x} = \frac{dv_x}{dt}$

$x(t_2) - x(t_1) = \int_{t_1}^{t_2} v_x(t) dt$ (b) Beschleunigung $a_x(t) = \frac{dv_x}{dt} = \lim_{t \rightarrow 0} \frac{v_x(t) - v_x(t_0)}{t - t_0} = \frac{v_x(t) - v_x(t_0)}{t - t_0} \cdot \frac{1}{t - t_0}$

a_y wird analog berechnet. Man benötigt außerdem die Position zu den drei Zeitpunkten t_1, t_2, t_3 .

Aufgabe 1.7

(a) Kosten für Agent 1 = $11 \cdot 100 \text{ Cent} + 1 \cdot 1 \text{ Cent} = 1,101 \text{ Cent}$.

Kosten für 2 = $0 \cdot 100 \text{ Cent} + 38 \cdot 1 \text{ Cent} = 38 \text{ Cent}$.

Daher spart Agent 2 $1,101 \text{ Cent} - 38 \text{ Cent} = 1,063 \text{ Cent}$. (b) Gewinn für

Agent 1 = $189 \cdot 100 \text{ Cent} + 799 \cdot 1 \text{ Cent} = 19,699 \text{ Cent}$.

Gewinn für Agent 2 = $200 \cdot 100 \text{ Cent} + 762 \cdot 1 \text{ Cent} = 20,762 \text{ Cent}$.

Agent 2 hat also $20,762 \text{ Cent} - 19,699 \text{ Cent} = 1,063 \text{ Cent}$ höheren Gewinn.

Bewertet man die durch Fehler entgangenen Gewinne, trifft der nutzenbasierte Agent die gleichen Entscheidungen wie ein kostenbasierter Agent.

11.2 Aussagenlogik

Aufgabe 2.1 Mit der Signatur $\Sigma = \{ \neg, \wedge, \vee, \rightarrow, \leftrightarrow \}$ und der Grammatikvariablenformel lässt sich die Syntax der Aussagenlogik wie folgt definieren:

Formel ::= $\neg \text{Formel} \mid \text{Formel} \wedge \text{Formel} \mid \text{Formel} \vee \text{Formel} \mid \text{Formel} \rightarrow \text{Formel} \mid \text{Formel} \leftrightarrow \text{Formel}$

$\neg \text{Formel} \mid \text{Formel} \wedge \text{Formel} \mid \text{Formel} \vee \text{Formel} \mid \text{Formel} \rightarrow \text{Formel} \mid \text{Formel} \leftrightarrow \text{Formel}$

$\text{Formel} \wedge \text{Formel} \mid \text{Formel} \vee \text{Formel} \mid \text{Formel} \rightarrow \text{Formel} \mid \text{Formel} \leftrightarrow \text{Formel}$

$\text{Formel} \rightarrow \text{Formel} \mid \text{Formel} \leftrightarrow \text{Formel}$

Aufgabe 2.2 Beweis mit der Wahrheitstabellenmethode.

Aufgabe 2.3 (a) $(\neg A \vee B) \vee (\neg B \vee A)$ (b) $(\neg A \vee B) \vee (\neg B \vee A)$ (c) t

Aufgabe 2.4 (a) erfüllbar (b) wahr (c) unerfüllbar

Aufgabe 2.6

(a) In Aufgabe 2.3(c) wurde bereits gezeigt, dass $A \vee (A \vee B) \vee B$ eine Tautologie ist.

Der Abzugssatz stellt somit die Richtigkeit der Inferenzregel sicher.

(b) Wir zeigen mit der Wahrheitstabellenmethode, dass $(A \vee B) \vee (\neg B \vee C) \vee (A \vee C)$ eine Tautologie.

Aufgabe 2.7 Die Anwendung der Resolutionsregel auf die Klausel $(f \vee B)$ und $(\neg B \vee f)$ ergibt das Resolvente $(f \vee f) \vee (f)$. Nun wenden wir die Auflösungsregel auf die Klauseln B und $\neg B$ an und erhalten die leere Klausel als Resolvente. Weil $(f \vee B) \vee B$ und $(\neg B \vee f) \vee \neg B$, $(f) \vee ()$. In der Praxis ist es wichtig, dass die Ableitung des Leersatzes immer auf einen Widerspruch zurückzuführen ist.

Aufgabe 2.8 Wenn KB einen Widerspruch enthält, dann gibt es zwei Klauseln A und $\neg A$, aus denen sich die leere Klausel ableiten lässt. Der Widerspruch in KB besteht eindeutig immer noch in $KB \vee \neg Q$. Daher ermöglicht es auch die Ableitung der Leerklausel.

Aufgabe 2.9 (a) $(A \vee B) \vee (\neg A \vee \neg B)$ (b) $(A \vee B) \vee (B \vee C) \vee (A \vee C)$

Übung 2.10 Formalisierung: Komplize: A, Auto: C, Schlüssel: K

$$WB \vee (A \vee C) \vee [(\neg A \vee \neg K) \vee (A \vee K)] \vee K$$

Transformation in CNF: $(\neg A \vee \neg K) \vee (A \vee K) \vee (\neg K \vee A) \vee (\neg A \vee K)$

Versuchen Sie, C zu beweisen und $\neg C$ zur Klauselmenge hinzuzufügen. Der CNF-Klauselsatz ist

$$(\neg A \vee C)1 \vee (\neg K \vee A)2 \vee (\neg A \vee K)3 \vee (K)4 \vee (\neg C)5.$$

Auflösungsbeweis : Res(2, 4) : (A)6

Res(1, 6) : (C)7

Res(7, 5) : ()8

Somit wurde C gezeigt.

Aufgabe 2.11

(a) $KB \vee (A \vee B) \vee (\neg B \vee C)$, $Q \vee (A \vee C)$

$$KB \vee \neg Q \vee (A \vee B)1 \vee (\neg B \vee C)2 \vee (\neg A)3 \vee (\neg C)4$$

Auflösungsnachweis : Res(1, 3): (B)5

Res(2, 4) : $(\neg B)6$

Res(5, 6) : ()

(b) $\neg(\neg B \vee (B \vee \neg A) \vee \neg A) \vee (\neg B)1 \vee (B \vee \neg A)2 \vee (A)3$

Auflösungsnachweis : Res(1, 2): $(\neg A)4$

Res(3, 4) : ()

Aufgabe 2.12 Durch Anwendung der Äquivalenzen aus Satz 2.1 auf Seite 18 können wir die Behauptungen sofort beweisen.

Übung 2.13

$$\text{Res}(8, 9) : (C \vee F \vee E \vee f)10$$

$$\text{Res}(3, 10) : (F \vee E \vee f)11$$

$$\text{Res}(6, 11) : (A \vee B \vee C \vee E \vee f)12$$

$$\text{Res}(1, 12) : (B \vee C \vee E \vee f)13$$

$$\text{Res}(2, 13) : (C \vee E \vee f)14$$

$$\text{Res}(3, 14) : (E \vee f)15$$

$$\text{Res}(5, 15) : ()$$

11.3 Prädikatenlogik erster Ordnung

Aufgabe 3.1

- (a) $\forall x \text{ männlich}(x) \rightarrow \neg \text{weiblich}(x)$
 (b) $\forall x \forall y \forall z \text{ Vater}(x, y) \rightarrow \text{männlich}(x) \rightarrow \text{Kind}(y, x, z)$ (c)
 $\forall x \forall y \text{ Geschwister}(x, y) \rightarrow [(\forall z \text{ Vater}(z, x) \rightarrow \text{Vater}(z, y)) \rightarrow (\forall z \text{ Mutter}(z, x) \rightarrow \text{Mutter}(z, y))]$
 (d) $\forall x \forall y \forall z \text{ Eltern}(x, y, z) \rightarrow \text{Vater}(x, z) \rightarrow \text{Mutter}(y, z)$ (e) $\forall x \forall y$
 $\text{Onkel}(x, y) \rightarrow \forall z \forall u \text{ Kind}(y, z, u) \rightarrow \text{Geschwister}(z, x) \rightarrow \text{männlich}(x)$ (f) $\forall x$
 $\forall y \text{ Vorfahr}(x, y) \rightarrow \forall z \text{ Kind}(y, x, z) \rightarrow \forall u \forall v \text{ kind}(u, x, v) \rightarrow$
 $\text{Vorfahr}(u, y))$

Aufgabe 3.2

- (a) $\forall x \forall y \forall z \text{ Vater}(y, x) \rightarrow \text{Mutter}(z, x)$ (b)
 $\forall x \forall y \text{ Kind}(y, x, z)$ (c)
 $\forall x \text{ Vogel}(x) \rightarrow \text{fliegt}(x)$ (d)
 $\forall x \forall y \forall z \text{ Tier}(x) \rightarrow \text{Tier}(y) \rightarrow \text{isst}(x, y) \rightarrow \text{isst}(y, z) \rightarrow \text{Getreide}(z)$ (e) $\forall x \text{ Tier}(x)$
 $\rightarrow (\forall y (\text{isst}(x, y) \rightarrow (\text{Pflanze}(y) \rightarrow (\text{Tier}(y) \rightarrow \forall z \text{ Pflanze}(z) \rightarrow \text{isst}(y, z) \rightarrow \text{viel\_kleiner}(y, x))))))$

Aufgabe 3.3 $\forall x \forall y \forall z x = \text{Vater}(y) \rightarrow \text{männlich}(x) \rightarrow \text{Kind}(y, x, z)$

Aufgabe 3.4

- $\forall x \forall y x < y \rightarrow y < x \rightarrow x = y$, $\forall x \forall y$
 $x < y \rightarrow \neg y < x$, $\forall x \forall y \forall z x < y$
 $\rightarrow y < z \rightarrow x < z$

Übung 3.5

- (a) MGU: $x/f(z), u/f(y)$, Term: $p(f(z), f(y))$ (b)
 nicht vereinbar
 (c) MGU: $x/\cos y, z/4 \rightarrow 7 \cdot \cos y$, Term: $\cos y = 4 \rightarrow 7 \cdot \cos y$
 (d) nicht
 vereinbar (e) MGU: $u/f(g(w, w), g(g(w, w), g(w, w))), g(g(g(w, w), g(w, w))), g(g(w, w), g(w, w)))$, $x/g(w, w), y/g(g(w, w), g(w, w)), z/g(g(g(w, w), g(w, w))), g(g(w, w), g(w, w)))$ Term:
 $q(f(g(w, w), g(g(w, w), g(w, w))), g(g(g(w, w), g(w, w))), g(g(w, w), g(w, w))), f(g(w, w), g(g(w, w), g(w, w))), g(g(g(w, w), g(w, w))), g(g(w, w), g(w, w)))$

Aufgabe 3.7

- (a) Gegeben sei die unerfüllbare Formel $p(x) \rightarrow \neg p(x) \rightarrow r(x)$. Wir wählen die Klausel $r(x)$ als SOS, sodass kein Widerspruch abgeleitet werden kann. (b) Wenn das SOS bereits unerfüllbar ist, kann kein Widerspruch abgeleitet werden. Wenn nicht, sind Auflösungsschritte zwischen Klauseln aus SOS und $(KB \rightarrow \neg Q)$ SOS erforderlich.

- (c) Wenn es kein Komplement zu einem Literal L in einer Klausel K gibt, dann verbleibt das Literal L in jeder Klausel, die durch Auflösung aus Klausel K abgeleitet wird. Somit kann die leere Klausel weder aus K oder seiner Auflösung abgeleitet werden, noch aus irgendeinem anderen zukünftige Entschlossenheit.

Aufgabe 3.8 $\neg Q \dot{\vee} KB \dot{\vee} (e = n)1 \dot{\vee} (n \cdot x = n)2 \dot{\vee} (e \cdot x = x)3 \dot{\vee} (\neg a = b)4$ Beweis: Dem(1, 2) : $(e \cdot x = e)5$ Tra,Sym(3, 5) : $(x = e)6$ Dem(4, 6) : $(\neg e = b)7$ Dem(7, 6) : $(\neg e = e)8$ ()

Dabei steht „Dem“ für Demodulation. Klausel Nummer 6 wurde durch Anwendung der Transitivität und Symmetrie der Gleichheit in den Klauseln 3 und 5 abgeleitet.

Übung 3.9 Die LOP-Eingabedateien sind:

(a) a;b<- . <- a,b. b;c<- . <- b,c. c;a<- . <-c,a.	(b) <- rasiert(barb,X), rasiert(X,X). rasiert(barb,X); rasiert(X,X) <-.	(c) e = n. <- a = b. m(m(X,Y),Z)=m(X,m(Y,Z)). m(e,X) = X. m(n,X) = n.

11.4 Grenzen der Logik

Aufgabe 4.1

- (a) Richtig: *Wir nehmen eine vollständige Beweisrechnung für PL1. Damit finden wir für jede wahre Formel in endlicher Zeit einen Beweis. Für alle unerfüllbaren Formeln gehe ich wie folgt vor: Ich wende den Kalkül auf $\neg\dot{\vee}$ an und zeige, dass $\neg\dot{\vee}$ wahr ist. Somit ist $\dot{\vee}$ falsch. Somit können wir jede wahre Formel aus PL1 beweisen und jede falsche Formel widerlegen.* Leider ist dieses Verfahren für erfüllbare Formeln ungeeignet.

Übung 4.2 (a)

Er rasiert sich genau dann, wenn er sich nicht rasiert. (Widerspruch) (b) Die Menge aller Mengen, die sich selbst nicht enthalten. Es enthält genau dann sich selbst wenn es sich nicht selbst enthält.

11.5 PROLOG

Übung 5.1 PROLOG signalisiert einen Stapelüberlauf. Der Grund ist die Tiefensuche von PROLOG, die immer das erste vereinbare Prädikat in der Eingabedatei wählt.

Bei rekursiven Prädikaten wie Gleichheit führt dies zu einer nicht terminierenden Rekursion.

Übung 5.2

```

write_path( [H1,H2|T] ) :- write_path([H2|T]), write_move(H2,H1). write_path( [_X] ).
write_move( state(X,W,Z,K),
state(Y,W,Z,K) ) :-
    write('Farmer from '), write(X), write(' to '), write(Y), nl.
write_move( state(X,X,Z,K), state(Y,Y,Z,K) ) :-
    write('Bauer und Wolf aus '), write(X), write(' bis '), write(Y), nl.
write_move( state(X,W,X,K), state(Y,W,Y,K) ) :-
    write('Bauer und Ziege von '), write(X), write(' bis '), write(Y), nl.
write_move( state(X,W,Z,X), state(Y,W,Z,Y) ) :-
    write('Bauer und Kohl aus '), write(X), write(' bis '), write(Y), nl.

```

Aufgabe 5.3

(a) Es wird benötigt, die gefundenen Lösungen auszugeben. Die Vereinheitlichung im Faktenplan (Ziel, Ziel, Pfad, Pfad). kümmert sich darum. (b)

Die Bedingungen für den Eintritt in die Klauseln des Prädikats sicher überschneiden sich. Das durch den Fehler verursachte Zurückverfolgen führt zur Ausführung der zweiten oder dritten Alternative von sicher, wobei die gleiche Lösung offensichtlich wieder gefunden wird. Eine Kürzung am Ende der ersten beiden Safe-Klauseln löst das Problem. Alternativ können alle sicheren Zustände explizit angegeben werden, wie zum Beispiel „safe(state(left,left,left,right))“.

Aufgabe 5.5 ?- eins(10). one(0) :-

```

write(1). eins(N) :- N1 ist N-1,
eins(N1), eins(N1).

```

Aufgabe 5.6

(a) Mit $n1 = |L1|$, $n2 = |L2|$ es ist wahr, dass $Tappend(n1, n2) = \check{y}(n1)$. (b)

Mit $n = |L|$ es gilt, dass $Tnrev(n) = \check{y}(n2)$. (c) Mit $n =$

$|L|$ es ist wahr, dass $Tacrev(n) = \check{y}(n)$.

Übung 5.7 Für die Bäume mit Symbolen auf den inneren Knoten können wir beispielsweise die Terme $a(b,c)$ und $a(b(e,f,g),c(h),d)$ verwenden. Bäume ohne Symbole: Baum(b,c) oder Baum(Baum(e,f,g),Baum(h),d).

Aufgabe 5.8 SWI-PROLOG-Programme:

(a) fib(0,1). fib(1,1).

fib(N,R):- N1 ist N-1, fib(N1,R1), N2 ist N-2, fib(N2,R2), R ist R1 + R2.

(b) T (n) = $\check{y}(2)$ (c) :- dynamische fib/2.

fib(0,1). fib(1,1). $\check{N}$.

fib(N,R) :- N1 ist N-1, fib(N1,R1), N2 ist N-2, fib(N2,R2), R ist R1 + R2,
Asserta(fib(N,R)).

(d) Wenn die Fakten fib(0,1) bis fib(k,fib(k)) bereits berechnet wurden, dann gilt für den Aufruf fib(n,X) das

$$T(n) = \begin{cases} \check{y}(1) & \text{wenn } n \check{y} k, \\ \check{y}(n \check{y} k), & \text{wenn } n > k. \end{cases}$$

- (e) Weil nach den ersten n Aufrufen des `fib`-Prädikats alle weiteren Aufrufe direkt sind Zugang zu Fakten.

Übung 5.9

- (a) Die Lösung wird hier nicht offengelegt.

(B)

Start :-

```
fd_domain([Brite, Schwede, Däne, Norwegisch, Deutsch],1,5),
fd_all_different([Brite, Schwede, Däne, Norweger, Deutsch]),
fd_domain([Tee, Kaffee, Wasser, Bier, Milch],1,5),
fd_all_different([Tee, Kaffee, Wasser, Bier, Milch]),
fd_domain([Rot, Weiß, Grün, Gelb, Blau],1,5),
fd_all_different([Rot, Weiß, Grün, Gelb, Blau]),
fd_domain([Hund, Vogel, Katze, Pferd, Fisch],1,5),
fd_all_different([Hund, Vogel, Katze, Pferd, Fisch]),
fd_domain([Pallmall, Dunhill, Marlboro, Winfield, Rothmanns],1,5),
fd_all_different([Pallmall, Dunhill, Marlboro, Winfield, Rothmanns]),
fd_labeling([Brite, Schwede, Däne, Norwegisch, Deutsch]),
Brite #= Rot, % Der Brite lebt im roten Haus
Schwede #= Hund, % Der Schwede hat keinen Hund
Däne #= Tee, % Der Däne trinkt gern Tee
Grün #= Weiß - 1, % Das grüne Haus befindet sich links vom weißen Haus
Grün #= Kaffee, % Der Besitzer des Gewächshauses trinkt Kaffee
Pallmall #= Bird, % Die Person, die Pall Mall raucht, hat einen Vogel
Milch #= 3, % Der Mann im mittleren Haus trinkt Milch
Gelb #= Dunhill, % Der Besitzer des gelben Hauses raucht Dunhill
Norweger #= 1, % Der Norweger wohnt im ersten Haus
dist(Marlboro,Cat)#=1, % Der Marlboro-Raucher wohnt neben der Katze
dist(Horse,Dunhill) #= 1, % Der Mann mit dem Pferd wohnt neben dem Dunhill-Raucher
Winfield #= Beer, % Der Winfield-Raucher trinkt gern Bier
dist(Norwegian,Blue) #= 1, % Der Norweger wohnt neben dem blauen Haus
Deutsch #= Rothmanns, % Der Deutsche raucht Rothmanns
dist(Marlboro,Water)#=1, % Der Nachbar des Marlboro-Rauchers trinkt Wasser.
write([Brite, Schwede, Däne, Norwegisch, Deutsch]), nl,
write([Hund, Vogel, Katze, Pferd, Fisch]), nl.
```

11.6 Suche, Spiele und Problemlösung

Übung 6.1

- (a) Auf der letzten Ebene gibt es b^D Knoten. Alle vorherigen Level zusammen haben

$$Nb(dmax) = \sum_{ich=0}^{d-1} \frac{b^{d-ich}}{b-1} = \frac{b^d - 1}{b - 1}$$

Knoten, wenn b groß wird. Weil $b^d / (b-1) \approx b^{d-1}$ gibt es ungefähr b mal so viele Knoten auf der letzten Ebene wie auf allen anderen Ebenen zusammen.

- (b) Für einen nicht konstanten Verzweigungsfaktor gilt diese Aussage nicht mehr, wie das folgende Gegenbeispiel zeigt: Ein Baum, der bis zur vorletzten Ebene stark verzweigt, gefolgt von einer Ebene mit einem konstanten Verzweigungsfaktor von 1, hat genau so viele Knoten auf der letzten Ebene wie auf der vorletzten Ebene.

Übung 6.2

- (a) In Abb. 6.3 auf Seite 86 die Struktur des Baums auf der zweiten Ebene mit seinen acht Knoten wiederholt sich. Somit können wir aus b den durchschnittlichen Verzweigungsfaktor bm berechnen $8 \text{ zu } bm = \sqrt[8]{8}$.

- (b) Hier ist die Berechnung nicht so einfach, da der Wurzelknoten des Baums stärker verzweigt als alle anderen. Wir können jedoch sagen, dass im Inneren des Baumes der Verzweigungsfaktor genau um 1 kleiner ist, als er ohne die Zyklusprüfung wäre. Somit ist $b^* \approx 8^{1/8} \approx 1,8$.

Aufgabe 6.3

- (a) Für den durchschnittlichen Verzweigungsfaktor ist die Anzahl der Blattknoten festgelegt. Beim effektiven Verzweigungsfaktor hingegen ist die Anzahl der Knoten im gesamten Baum festgelegt.
- (b) Weil die Anzahl aller Knoten im Baum normalerweise ein besseres Maß für die Rechenzeit für die Suche eines gesamten Baums ist als die Anzahl der Blattknoten. (c) Für ein großes n gilt nach (6.1) auf Seite 87 $n \approx b^* d + 1 / b^* = b^* d$, ergibt $b^* = \sqrt[n]{d}$.

Aufgabe 6.4

- (a) 3-Puzzle: $4! = 24$ Staaten, 8-Puzzle: $9! = 362\,880$ Staaten, 15-Puzzle: $16! = 20\,922\,789\,888\,000$ Staaten.
- (b) Nachdem wir das leere Quadrat 12-mal im Uhrzeigersinn verschoben haben, erreichen wir wieder den Ausgangszustand und erzeugen so einen zyklischen Unterraum mit 12 Zuständen.

Aufgabe 6.6 (a) Mathematica-Programm:

```

0 BreadthFirstSearch[Node_, Goal_, Depth_] := Module[{i, NewNodes={}},
1   For[i=1, i<= Länge[Knoten], i++, 2
2     NewNodes = Join[ NewNodes, Successors[ Node[[i]] ] ];
3     If[MemberQ[Successors[ Node[[i]] ], Goal],
4       Return[{"Lösung gefunden", Tiefe+1}], 0
5     ]
6   ];
7   Wenn[Länge[NeueNodes] > 0, 8
8     BreadthFirstSearch[NewNodes, Ziel, Tiefe+1],
9     Return[{"Fail", Depth}]
10  ]
11]

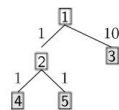
```

- (b) Weil die Tiefe des Suchraums nicht begrenzt ist.

Aufgabe 6.7

- (a) Für konstante Kosten 1 sind die Kosten aller Pfade in der Tiefe d kleiner als die Kosten aller Pfade in der Tiefe $d + 1$. Da alle Knoten in der Tiefe d vor dem ersten Knoten in der Tiefe $d + 1$ getestet werden, Es wird garantiert, dass eine Lösung der Länge d vor einer Lösung der Länge $d + 1$ gefunden

- wird. (b) Für den benachbarten Suchbaum werden zuerst Knoten Nummer 2 und der Lösungsknoten Nummer 3 mit Kosten 10 generiert. Die Suche wird beendet. Die Knoten 4 und 5 mit Pfadkosten von jeweils 2 werden nicht generiert. Die optimale Lösung wird daher nicht gefunden.



$$\begin{aligned}
 P(Klasse = \text{gut} \mid y \neq 3) &= \frac{P(y \neq 3 \mid Klasse = \text{gut}) \cdot P(Klasse = \text{gut})}{P(y \neq 3)} \\
 &= \frac{2/9 \cdot 9/13}{6/13} = \frac{1}{3}.
 \end{aligned}$$

(b) $P(y > 3 \mid Klasse = \text{gut}) = 7/9$.

Aufgabe 7.3

(a) acht

Ereignisse. (B)

$$\begin{aligned}
 &P(Prec = \text{trocken} \mid \text{Himmel} = \text{klar}, \text{Balken} = \text{steigend}) \\
 &= \frac{P(Prec = \text{trocken}, \text{Himmel} = \text{klar}, \text{Balken} = \text{steigend})}{P(\text{Himmel} = \text{klar}, \text{Balken} = \text{steigend})} = \frac{0,4}{0,47} = 0,85.
 \end{aligned}$$

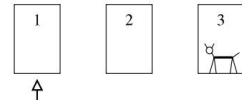
(c) Fehlende Zeile in der Tabelle: Bewölkt Fallend Regen 0,12

$$\begin{aligned}
 P(Prec = \text{regnet} \mid \text{Sky} = \text{bewölkt}) &= \frac{P(Prec = \text{regnet}, \text{Sky} = \text{bewölkt})}{P(\text{Himmel} = \text{bewölkt})} \\
 &= \frac{0,23}{0,35} = 0,66.
 \end{aligned}$$

(d) Das fehlende Wahrscheinlichkeitsmaß beträgt 0,15. Das Indifferenzprinzip (Definition 7.5 auf Seite 126) erfordert nun, dass beiden fehlenden Zeilen symmetrisch der Wert 0,075 zugewiesen wird.

Übung 7.4

Der Kandidat wählte zunächst Tür Nummer 1. Dann öffnete der Moderator Tür Nummer 3. Wir führen die Abkürzung A_i für „Auto steht hinter Tür Nummer i (bevor das Experiment beginnt)“ und M_i für „Gastgeber (Moderator) öffnet Tür Nummer i “ ein. Dann ist $P(A_1) = P(A_2) = P(A_3) = 1/3$ und wir berechnen



$$\begin{aligned}
 P(A_1 \mid M_3) &= \frac{P(M_3 \mid A_1)P(A_1)}{P(M_3)} \\
 &= \frac{P(M_3 \mid A_1)P(A_1)}{P(M_3 \mid A_1)P(A_1) + P(M_3 \mid A_2)P(A_2) + P(M_3 \mid A_3)P(A_3)} = \frac{1/2 \cdot 1/3}{1/2 \cdot 1/3 + 1 \cdot 1/3 + 0 \cdot 1/3} = \frac{1/2 \cdot 1/3}{2/3} = 1/4 \\
 P(A_2 \mid M_3) &= \frac{P(M_3 \mid A_2)P(A_2)}{P(M_3)} = \frac{1 \cdot 1/3}{2/3} = 1/2
 \end{aligned}$$

Somit ist klar, dass es besser ist, zur anderen Tür zu wechseln.

Aufgabe 7.5 Die Lagrange-Funktion lautet $L = \sum_{i=1}^n p_i \ln p_i + \lambda (\sum_{i=1}^n p_i - 1)$. Wenn man die partiellen Ableitungen nach p_i und λ gleich Null setzt, erhält man

$$\frac{\partial L}{\partial p_i} = \ln p_i + 1 + \lambda = 0 \text{ und } \frac{\partial L}{\partial \lambda} = \sum_{i=1}^n p_i - 1 = 0.$$

Die Subtraktion dieser beiden Gleichungen ergibt $p_i = p_j$ für alle $i, j \in \{1, \dots, n\}$. Somit ist $p_1 = p_2 = \dots = p_n = 1/n$.

Übung 7.6

PIT-

Eingabedaten: var A{t,f},

B{t,f}; P([A=t]) = 0,5;

P([B=t] | [A=t]) = 0,94; QP([B=t]);

PIT-Ausgabe:

Status der Abfragen melden

Nr Wahrheitswert-Wahrscheinlichkeitsabfrage
1 NICHT ANGEGBEN 7.200e-01 QP([B=t]);

Da PIT numerisch arbeitet, kann es mit dem λ nicht symbolisch rechnen. Parameter λ und λ , eher nur mit konkreten Zahlen.

Übung 7.7

Wenn: $p_1 = P(A, B)$, $p_2 = P(A, \neg B)$, $p_3 = P(\neg A, B)$, $p_4 = P(\neg A, \neg B)$

$$\text{Die Einschränkungen sind: } p_1 + p_2 = \bar{y} \quad (11.1)$$

$$p_4 = 1 - \bar{y} \quad (11.2)$$

$$p_1 + p_2 + p_3 + p_4 = 1 \quad (11.3)$$

Daraus folgt $p_3 = \bar{y} - \bar{y}$. Aus (11.1) schließen wir auf der Grundlage der Indifferenz, dass $p_1 = p_2 = \bar{y}/2$. Somit ist $P(B) = p_1 + p_3 = \bar{y}/2 + \bar{y} - \bar{y} = \bar{y}/2 = P(A \bar{y} B) - \bar{y}/2P(A)$. Das entsprechende PIT-Programm liest

var A{t,f}, B{t,f}; P([A=t]) = 0,6;

P([B=t] ODER [A=t]) =

0,94; QP([B=t]).

Aufgabe 7.8 Die Lagrange-Funktion lautet

$$L = \sum_{i=1}^4 p_i \ln p_i + \bar{y}_1(p_1 + p_2 - \bar{y}) + \bar{y}_2(p_1 + p_3 - \bar{y}) + \bar{y}_3(p_1 + p_2 + p_3 + p_4 - 1). \quad (11.4)$$

Durch partielle Ableitungen für p_1 , p_2 , p_3 , p_4 erhalten wir

$$\frac{\partial L}{\partial p_1} = \ln p_1 + \bar{y}_1 + \bar{y}_2 + \bar{y}_3 = 0, \quad \bar{y}_{p1} \quad (11.5)$$

$$\frac{\tilde{y}^L}{\tilde{y}^L} = \tilde{y}^L \ln p_2 \tilde{y}^L 1 + \tilde{y}^L 1 + \tilde{y}^L 3 = 0, \tilde{y}^L p_2 \quad (11.6)$$

$$\frac{\tilde{y}^L}{\tilde{y}^L} = \tilde{y}^L \ln p_3 \tilde{y}^L 1 + \tilde{y}^L 2 + \tilde{y}^L 3 = 0, \tilde{y}^L p_3 \quad (11.7)$$

$$\frac{\tilde{y}^L}{\tilde{y}^L} = \tilde{y}^L \ln p_4 \tilde{y}^L 1 + \tilde{y}^L 3 = 0 \tilde{y}^L p_4 \quad (11.8)$$

und berechnen

$$(11.5)-(11.6): \ln p_2 \tilde{y}^L \ln p_1 + \tilde{y}^L 2 = 0, \quad (11.9)$$

$$(11.7)-(11.8): \ln p_4 \tilde{y}^L \ln p_3 + \tilde{y}^L 2 = 0, \quad (11.10)$$

woraus folgt, dass $p_3/p_4 = p_1/p_2$, was wir sofort in (7.12) auf Seite 129 einsetzen:

$$\frac{p_2 p_3}{s.4} + p_2 + p_3 + p_4 = 1$$

$$\tilde{y}^L p_2 1 + p_4 \frac{p_3}{s.4} + p_3 + p_4 = 1 \quad (11.11)$$

$$(7.10)-(7.11): p_2 = p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L$$

was, ersetzt durch p_2 , $(p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L)(1 + \text{ergibt } \frac{p_3}{s.4}) + p_3 + p_4 = 1$

$$(7.10) \text{ in } (7.12): \tilde{y}^L + p_3 + p_4 = 1. \quad (11.12)$$

Daher eliminieren wir p_4 in (11.8), was ergibt

$$p_3 (p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L) 1 + \frac{1}{1 \tilde{y}^L \tilde{y}^L p_3} + 1 \tilde{y}^L \tilde{y}^L = 1 \quad (11.13)$$

$$\tilde{y}^L (p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L) (1 \tilde{y}^L \tilde{y}^L p_3 + p_3) = \tilde{y}^L (1 \tilde{y}^L \tilde{y}^L p_3) \quad (11.14)$$

$$\tilde{y}^L (p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L) (1 \tilde{y}^L \tilde{y}^L) = \tilde{y}^L (1 \tilde{y}^L \tilde{y}^L p_3) \quad (11.15)$$

$$\tilde{y}^L p_3 + \tilde{y}^L \tilde{y}^L \tilde{y}^L \tilde{y}^L p_3 \tilde{y}^L \tilde{y}^L 2 + \tilde{y}^L \tilde{y}^L = \tilde{y}^L \tilde{y}^L \tilde{y}^L 2 \tilde{y}^L \tilde{y}^L p_3 \quad (11.16)$$

$$\tilde{y}^L p_3 = \tilde{y}^L (1 \tilde{y}^L \tilde{y}^L). \quad (11.17)$$

Mit (11.12) folgt $p_4 = (1 \tilde{y}^L \tilde{y}^L)(1 \tilde{y}^L \tilde{y}^L)$ und mit (11.11) erhalten wir $p_2 = \tilde{y}^L (1 \tilde{y}^L \tilde{y}^L)$ und $p_1 = \tilde{y}^L \tilde{y}^L$.

Aufgabe 7.9

(a)

$$M = \begin{pmatrix} 0 & 1000 & 10 \\ 0 & & \end{pmatrix}.$$

(B)

$$M \cdot \begin{pmatrix} P \\ 1 \tilde{y}^L S \end{pmatrix} = \begin{pmatrix} 1000 \cdot (1 \tilde{y}^L p) \\ 10 \cdot p \end{pmatrix}.$$

Da die Entscheidung zwischen *Löschen* oder *Lesen* durch die Festlegung des Minimums der beiden Werte getroffen wird, reicht es aus, die beiden Werte zu vergleichen: $1000(1 - p) < 10p$, was $p > 0,99$ ergibt. Im Allgemeinen berechnen wir für eine Matrix M den Schwellenwert $k/(k+1)$.

Aufgabe 7.10

(a) Da A und B unabhängig sind, gilt $P(A|B) = P(A) = 0,2$. (b) Durch Anwendung der Konditionierungsregel (Seite 155) erhalten wir

$$P(C|A) = P(C|A, B)P(B) + P(C|A, \neg B)P(\neg B) = 0,2 \cdot 0,9 + 0,1 \cdot 0,1 = 0,19.$$

Aufgabe 7.11

(a)

$$\begin{aligned} P(AI) &= P(AI, Bur, Ohr) + P(AI, \neg Bur, Ohr) \\ &\quad + P(AI, Bur, \neg Ohr) + P(AI, \neg Bur, \neg Ohr) \\ &= P(AI|Bur, Ohr)P(Bur, Ohr) + P(AI|\neg Bur, Ohr)P(\neg Bur, Ohr) \\ &\quad + P(AI|Bur, \neg Ohr)P(Bur, \neg Ohr) \\ &\quad + P(AI|\neg Bur, \neg Ohr)P(\neg Bur, \neg Ohr) \\ &= 0,95 \cdot 0,001 \cdot 0,002 + 0,29 \cdot 0,999 \cdot 0,002 + 0,94 \cdot 0,001 \cdot 0,998 \\ &\quad + 0,001 \cdot 0,999 \cdot 0,998 \\ &= 0,00252, \\ P(J) &= P(J, AI) + P(J, \neg AI) = P(J|AI)P(AI) + P(J|\neg AI)P(\neg AI) \\ &= 0,9 \cdot 0,0025 + 0,05 \cdot (1 - 0,0025) = 0,052, \\ P(M) &= P(M, AI) + P(M, \neg AI) = P(M|AI)P(AI) + P(M|\neg AI)P(\neg AI) = 0,7 \cdot 0,0025 \\ &\quad + 0,01 \cdot (1 - 0,0025) = 0,0117. \end{aligned}$$

(B)

$$\begin{aligned} P(AI|Bur) &= P(AI|Bur, Ohr)P(Ohr) + P(AI|Bur, \neg Ohr)P(\neg Ohr) \\ &= 0,95 \cdot 0,002 + 0,94 \cdot 0,998 = 0,94002, \\ P(M|Bur) &= P(M, Bur)/P(Bur) = [P(M, AI, Bur) + P(M, \neg AI, Bur)] \\ &\quad /P(Bürer) \\ &= [P(M|AI)P(AI|Bur)P(Bur) + P(M|\neg AI)P(\neg AI|Bur)P(Bur)] \\ &\quad /P(Bürer) \\ &= P(M|AI)P(AI|Bur) + P(M|\neg AI)P(\neg AI|Bur) \\ &= 0,7 \cdot 0,94002 + 0,01 \cdot 0,05998 = 0,659. \end{aligned}$$

(C)

$$P(Bur|M) = \frac{P(M|Bur)P(Bur)}{P(M)} = \frac{0,659 \cdot 0,001}{0,0117} = 0,056.$$

(D)

$$\begin{aligned}
 P(A|J, M) &= \frac{P(A, J, M)}{P(J, M)} = \frac{P(A, J, M)}{P(A, J, M) + P(\neg A, J, M)} \\
 &= \frac{1}{1 + \frac{P(\neg A, J, M)}{P(A, J, M)}} = \frac{1}{\frac{P(J|\neg A)P(M|\neg A)P(\neg A)1 + P(J|A)P(M|A)P(A)}{P(A, J, M)}} \\
 &= \frac{1}{1 + \frac{0,05 \cdot 0,01 \cdot 0,9975}{0,9 \cdot 0,7 \cdot 0,0025}} = 0,761,
 \end{aligned}$$

$$\begin{aligned}
 P(J, M|Bur) &= P(J, M|A)P(A|Bur) + P(J, M|\neg A)P(\neg A|Bur) \\
 &= P(J|A)P(M|A)P(A|Bur) \\
 &\quad + P(J|\neg A)P(M|\neg A)P(\neg A|Bur) \\
 &= 0,9 \cdot 0,7 \cdot 0,94 + 0,05 \cdot 0,01 \cdot 0,06 = 0,5922,
 \end{aligned}$$

$$\begin{aligned}
 P(A|\neg Bur) &= P(A|\neg Bur, Ohr)P(Ohr) + P(A|\neg Bur, \neg Ohr)P(\neg Ohr) \\
 &= 0,29 \cdot 0,002 + 0,001 \cdot 0,998 = 0,00158,
 \end{aligned}$$

$$\begin{aligned}
 P(J, M|\neg Bur) &= P(J, M|A)P(A|\neg Bur) + P(J, M|\neg A)P(\neg A|\neg Bur) \\
 &= P(J|A)P(M|A)P(A|\neg Bur) \\
 &\quad + P(J|\neg A)P(M|\neg A)P(\neg A|\neg Bur) \\
 &= 0,9 \cdot 0,7 \cdot 0,00158 + 0,05 \cdot 0,01 \cdot 0,998 = 0,00149,
 \end{aligned}$$

$$\begin{aligned}
 P(J, M) &= P(J, M|Bur)P(Bur) + P(J, M|\neg Bur)P(\neg Bur) \\
 &= 0,5922 \cdot 0,001 + 0,00149 \cdot 0,999 = 0,00208, P(J,
 \end{aligned}$$

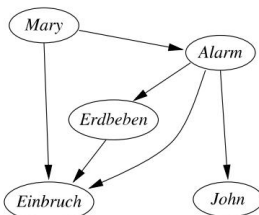
$$P(Bur|J, M) = \frac{P(J, M|Bur)P(Bur)}{P(J, M)} = \frac{0,5922 \cdot 0,001}{0,00208} = 0,284.$$

(e) $P(J)P(M) = 0,052 \cdot 0,0117 = 0,00061$, aber $P(J, M) = 0,00208$ (siehe oben).

Daher ist $P(J)P(M) \neq P(J, M)$.

(f) Siehe Abschn. 7.4.5 auf Seite 149.

(G)



Aufgrund der kontraproduktiven Variablenordnung sind auch andere Lösungen möglich. So

lernen wir: Verwenden Sie immer

eine Variablenreihenfolge, die die Kausalität berücksichtigt!

(h) Nur die CPT des

Alarmknotens ändert sich.
Alle anderen Knoten

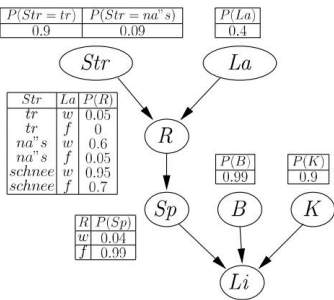
sind erdbebenunabhängig
oder bei Erdbebenalarm
unabhängig.

(i) $Bur P(A)$

T	0,94

f	0,0016

Abb. 11.1 Bayesianisches Netzwerk für das Fahrradlicht.
Die CPT für Li ist in der Aufgabe angegeben
Stellungnahme



Aufgabe 7.12

- (a), (b) Siehe Abb. 11.1.
- (c) Die Kante (Str, Li) fehlt, weil Li und Str bei gegebenem V bedingt unabhängig sind, also $P(Li|V, Str) = P(Li|V)$, weil der Straßenzustand keinen direkten Einfluss auf das Licht hat, sondern nur über den Dynamo und die von ihm erzeugte Spannung. (Die bedingte Unabhängigkeit Li und Str bei gegebenem V kann hier nicht durch Berechnung anhand der verfügbaren Daten für $P(Li|V, Str)$ und $P(Li|V)$ gezeigt, sondern nur behauptet werden.)
- (d) Für das Gegebene CPTs:

$$\begin{aligned} P(R|Str) &= P(R|Str,Flw)P(Flw) + P(R|Str,\neg Flw)P(\neg Flw) \\ &= 0,95 \cdot 0,4 + 0,7 \cdot 0,6 = 0,8, \\ P(V|Str) &= P(V|R)P(R|Str) + P(V|\neg R)P(\neg R|Str) \\ &= 0,04 \cdot 0,8 + 0,99 \cdot 0,2 = 0,23. \end{aligned}$$

11.8 Maschinelles Lernen und Data Mining

Aufgabe 8.1

- (a) Der Agent ist eine Funktion A, die den Vektor (t1,t2,t3, d1, d2, d3, f1, f2, f3) bestehend aus jeweils drei Werten für Temperatur, Druck und Feuchtigkeit einer Klasse zuordnet Wert k $\in$ {sonnig, bewölkt, regnerisch}. (b) Die Trainingsdatendatei besteht aus einer Zeile des Formulars

t1,t2,t3, d1, d2, d3, f1, f2, f3, ki

für jedes einzelne Trainingsdatenelement. Der Index läuft über alle Trainingsdaten. Eine konkrete Datei könnte zum Beispiel beginnen:

17,17,17,980,983,986,63,61,50,sonnig
20,22,25,990,1014,1053,65,66,69, sonnig
20,22,18,990,979,929,52,60,61,regnerisch

Übung 8.2 Wir können die Symmetrie direkt anhand der Definition von Korrelationskoeffizienten oder der Kovarianz erkennen. Das Vertauschen von i und j ändert den Wert nicht. Für die

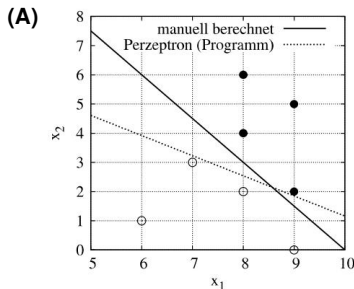
Diagonalelemente berechnen wir

$$K_{ii} = \frac{\bar{y}_{ii}}{s_i \cdot s_i} = \frac{\sum_{p=1}^N (x_p - \bar{x}_i)(x_p - \bar{x}_i)}{\sum_{p=1}^N (x_p - \bar{x}_i)^2} = 1.$$

Aufgabe 8.3 Die Wertefolge für w ist

(1, 1), (2,2, $\bar{y}_{0,4}$), (1,8, 0,6), (1,4, 1,6), (2,6, 0,2), (2,2, 1,2)

Übung 8.4



(b) Das Zeichnen einer geraden Linie im Diagramm ergibt:

$$1,5x_1 + x_2 - 15 > 0.$$

Nach 442 Iterationen sind die Perzeptron-Lernalgorithmen mit dem Startvektor $w = (1, 1, 1)$ kehrt zurück:

$$w = (11, 16, \bar{y}_{129}).$$

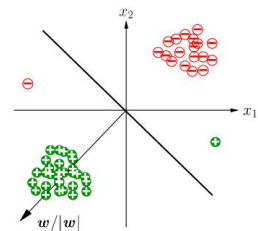
Dies entspricht: $0,69x_1 + x_2 - 8,06 > 0$

Aufgabe 8.5

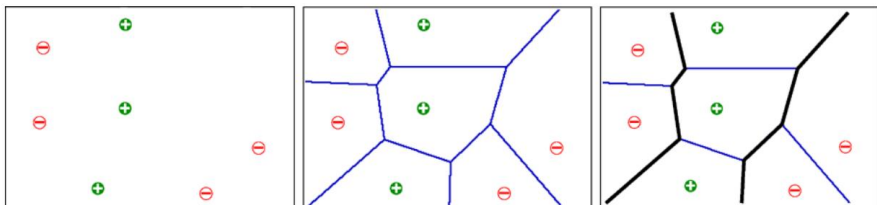
(a) Der Vektor $\bar{x}$ zeigt in eine „durchschnittliche“ Richtung aller positiven

Punkte und $\bar{x}_M + \bar{x}$ zeigt in eine „durchschnittliche“ Richtung aller negativen Punkte. Die Differenz dieser Vektoren zeigt von den negativen Punkten zu den positiven Punkten. Die Trennlinie steht dann senkrecht auf diesem Differenzvektor.

(b) Bei der Berechnung von w dominiert die Punktwolke der positiven und negativen Punkte. Die beiden Ausreißer spielen hier kaum eine Rolle. Für die Bestimmung der Trennlinie sind sie jedoch wichtig.



Übung 8.6



Aufgabe 8.7

- (a) Nächster Nachbar ist (8, 4), also Klasse 1.
 (b) $k = 2$: Klasse undefiniert, da eine Instanz von Klasse 1 und eine Instanz von Klasse 2.
 $k = 3$: Entscheidung 2:1 für Klasse
 $0. k = 5$: Entscheidung 2:3 für Klasse 1.

Aufgabe 8.8

- (a) Um allgemeine Aussagen treffen zu können, müssen wir davon ausgehen, dass die Punkte gleichmäßig im Merkmalsraum verteilt sind, das heißt, dass die Anzahl der Punkte pro Fläche insgesamt annähernd gleich ist. Nun berechnen wir die Fläche A_d eines schmalen Reihenrings der Breite Δd und des Abstands d um den Punkt x :

$$A_d = \pi(d + \Delta d)^2 - \pi d^2 = \pi(d^2 + 2d\Delta d + \Delta d^2) - \pi d^2 = 2\pi d\Delta d + \pi \Delta d^2.$$

- Das Gesamtgewicht aller Punkte im Ring der Breite Δd und des Abstands d ist somit proportional zu $d w_i = d / (1 + \Delta d^2) \approx 1 / (\Delta d)$ für $d \gg \Delta d$. (b) Für diese Gewichtung ist das Gesamtgewicht aller Punkte im Ring der Breite Δd und $d = d / \Delta d =$ Abstand d wäre proportional zu $d w_i$ 1. Somit hätte jeder Ring der Breite Δd das gleiche Gewicht, unabhängig von seinem Abstand zum Punkt x . Dies ist sicherlich nicht sinnvoll, da die unmittelbare Umgebung für die Annäherung am wichtigsten ist.

- Aufgabe 8.9 $\lim_{x \rightarrow 0} x \log_2 x = \lim_{x \rightarrow 0} x \log_2 x = \lim_{x \rightarrow 0} \frac{x \log_2 x}{1/x} = \lim_{x \rightarrow 0} \frac{\log_2 x}{1/x^2} = \lim_{x \rightarrow 0} \frac{\frac{1}{x \ln 2}}{-2/x^3} = \lim_{x \rightarrow 0} \frac{-x^2}{2 \ln 2} = 0$
 Gleichung die l'Hospitalsche Regel verwendet wurde.

- Aufgabe 8.10 (a) 0 (b) 1 (c) 1,5 (d) 1,875 (e) 2,32 (f) 2

Aufgabe 8.11

- (a) Aus $\log_2 x = \ln x / \ln 2$ folgt $c = 1 / \ln 2 \approx 1,44$. (b) Da sich beide Entropiefunktionen nur um einen konstanten Faktor unterscheiden, ändert sich die Lage des Entropiemaximums nicht. Extrema unter Einschränkungen behalten ebenfalls die gleiche Position. Somit stellt der Faktor c für die Max-Ent-Methode kein Problem dar. Beim Lernen von Entscheidungsbäumen werden die Informationsgewinne verschiedener Attribute verglichen. Da hier nur die Reihenfolge zählt, nicht aber der absolute Wert, stellt der Faktor c auch hier kein Problem dar.

Aufgabe 8.12

- (a) Für das erste Attribut berechnen wir

$$G(D, x_1) = H(D) - \sum_{i=1}^9 \frac{|D_{x_1=i}|}{|D|} H(D_{x_1=i})$$

$$= 1 - \frac{1}{8} H(D_{x_1=6}) + \frac{1}{8} H(D_{x_1=7}) + \frac{3}{8} H(D_{x_1=8}) + \frac{3}{8} H(D_{x_1=9})$$

$$+ \cdot 0,918 + \cdot 0,918 = 0,311 \frac{3}{8} -$$

$G(D, x_2) = 0,75$, also ist x_2 ausgewählt. Für $x_2 = 0, 1, 3, 4, 5, 6$ ist die Entscheidung klar. Für $x_2 = 2$ wird x_1 ausgewählt. Der Baum hat dann die Form

$x_2 = 0: 0 \ (1/0) \ x_2 = 1: 0$

$(1/0)$

$x_2 = 2:$

$| \ x_1 = 6: 0 \ (0/0) \ | \ x_1 = 7: 0$

$(0/0) \ | \ x_1 = 8: 0 \ (1/0) \ | \ x_1 =$

$9: 1 \ (1/0) \ x_2 = 3: 0 \ (1/0) \ x_2$

$= 4: 1 \ (1/0) \ x_2 = 5: 1 \ (1/0)$

$x_2 = 6: 1 \ (1/0)$

(b) Informationsgewinn für das kontinuierliche Attribut x_2 als Wurzelknoten:

Schwelle $\ddot{y}$ 012345

$G(D, x_2 \ddot{y})$ 0,138 0,311 0,189 0,549 0,311 0,138

Da $G(D, x_2 \ddot{y} 3) = 0,549 > 0,311 = G(D, x_1)$, wird $x_2 \ddot{y} 3$ ausgewählt. Für $x_2 \ddot{y} 3$ ist die Klassifikation nicht eindeutig. Wir berechnen $G(Dx_2 \ddot{y} 3, x_1) = 0,322$, $G(Dx_2 \ddot{y} 3, x_2 \ddot{y} 0) = 0,073$, $G(Dx_2 \ddot{y} 3, x_2 \ddot{y} 1) = 0,17$, $G(Dx_2 \ddot{y} 3, x_2 \ddot{y} 2) = 0,073$. Somit ist x_1 ausgewählt. Für $x_1 = 9$ ist die Klassifikation nicht eindeutig. Hier ist die Entscheidung eindeutig: $G(D(x_2 \ddot{y} 3, x_1=9), x_1) = 0$, $G(D(x_2 \ddot{y} 3, x_1=9), x_2 \ddot{y} 1) = 1$, $x_2 \ddot{y} 1$ wird gewählt, und Der Baum ist wieder zu 100 % korrekt.

Baum: x_2

$\leq 3: | \ x_1 = 6: 0 \ (1/0) \ | \ x_1$

$= 7: 0 \ (0/0) \ | \ x_1 = 8: 0 \ (1/0)$

$| \ x_1 = 9: | \ x_2 \leq 1: 0 \ (1/0)$

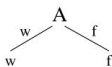
$| \ x_2 > 1: 1$

$(1/0) \ x_2 > 3: 1 \ (3/0)$

Übung 8.13

(a) 100 % für die Trainingsdaten, 75 % für die Testdaten, 80 % Gesamtkorrektheit.

(b) $(A \ddot{y} \neg C) \ddot{y} (\neg A \ddot{y} B)$ (c)



66,6 % Korrektheit für die Trainingsdaten,
100 % Korrektheit für die Testdaten,
80 % Gesamtkorrektheit

Aufgabe 8.14

- (a) Gleichung (8.7) auf Seite 189 zur Berechnung des Informationsgewinns eines Attributs A liest

$$\text{InfoGain}(D, A) = H(D) - \sum_{i=1}^N \frac{|D_i|}{|D|} H(D_i),$$

Dabei ist n die Anzahl der Werte des aktuell beobachteten Attributs. Wenn das Attribut A irgendwo im Teilbaum als Nachfolger des Wertes a_j getestet wird, dann kommt nur der Wert $A = a_j$ im Datensatz D_j und jeder seiner Teilmengen $D_{j,k}$ vor. Also $D_{j,k} = D_j$ und für alle $k \neq j$, $|D_{j,k}| = 0$ und wir erhalten

$$\text{InfoGain}(D_j, A) = H(D_j) - \sum_{i=1}^N \frac{|D_{j,i}|}{|D_j|} H(D_{j,i}) = H(D_j) - \sum_{i=1}^N \frac{|D_j|}{|D_j|} \frac{|D_j|}{|D_j|} H(D_j) = 0.$$

Das wiederholte Attribut hat somit einen Informationsgewinn von Null, weshalb es nicht mehr

verwendet wird. (b) Aus jedem kontinuierlichen Attribut A wird ein binäres Attribut $A > \tilde{D}$ generiert. Wenn weiter unten im Baum das Attribut A erneut mit einem anderen Schwellenwert $\tilde{D}'$ diskretisiert wird, dann unterscheidet sich das Attribut $A > \tilde{D}'$ von $A > \tilde{D}$. Hat es dann einen höheren Informationsgewinn als alle anderen Attribute, wird es im Baum verwendet. Dies bedeutet jedoch auch, dass bei mehreren Vorkommen eines kontinuierlichen Attributs die Schwellenwerte unterschiedlich sein müssen.

Übung 8.15

- (a) $P(\text{Himmel} = \text{klar}) = 0,65$
 $(\text{Balken} = \text{steigend}) = 0,67$

Himmel	Bar	P (Prec = dry Sky, Bar)
Klar	Steigend	0,85
Klar	Fallend	0,44
Bewölkt	Fallend	0,45
Bewölkt	Fallend	0,2

- (b) $P(\text{Himmel} = \text{klar}) = 0,65$

Bar	P (Prec = dry Bar)
Steigend	0,73
Fallend	0,33

Himmel	P (Balken = steigend Himmel)
Klar	0,72
Bewölkt	0,57

- (c) Die notwendigen CPTs für $P(\text{Prec}/\text{Sky}, \text{Bar})$ und $P(\text{Bar}/\text{Sky})$ sowie $P(\text{Sky})$ sind bereits bekannt.
 (d) $P_a = (0,37, 0,065, 0,095, 0,12, 0,11, 0,13, 0,023, 0,092)$
 $P_b = (0,34, 0,13, 0,06, 0,12, 0,15, 0,054, 0,05, 0,1)$
 $P = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$ (ursprüngliche Verteilung) Quadratisch

Abstand: $d_q(P_a, P) = 0,0029$, $d_q(P_b, P) = 0,014$ Kullback-Leibler-Abstand: $d_k(P_a, P) = 0,017$, $d_k(P_b, P) = 0,09$ Beide

Abstandsmetriken zeigen, dass das Netzwerk (a) nähert sich der Verteilung besser an als Netzwerk (b). Dies bedeutet, dass die Annahme, dass *Prec* und *Sky* bei gegebenem *Bar* bedingt unabhängig sind, weniger wahrscheinlich ist als die Annahme, dass *Sky* und *Bar* unabhängig sind.

(e) $P_c = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$. Diese Verteilung ist genau gleich der ursprünglichen Verteilung P . Dies ist nicht überraschend, da im Netzwerk keine Kanten fehlen. Dies bedeutet, dass keine Unabhängigkeiten angenommen wurden.

Übung 8.16 Wir können sofort erkennen, dass Scores und Perzeptrone äquivalent sind, indem wir ihre Definitionen vergleichen. Nun zur Äquivalenz zu naivem Bayes: Zuerst stellen wir fest, dass $P(K|S_1, \dots, S_n) > 1/2$ äquivalent zu $P(K|S_1, \dots, S_n) > P(\neg K|S_1, \dots, S_n)$, weil $P(\neg K|S_1, \dots, S_n) > 1 - P(K|S_1, \dots, S_n)$. Tatsächlich haben wir es hier mit einem binären naiven Bayes-Klassifikator zu tun.

Wir wenden den Logarithmus auf die naive Bayes-Formel an

$$P(K|S_1, \dots, S_n) = P \frac{P(S_1|K) \cdots P(S_n|K) \cdot P(K)}{(S_1, \dots, S_n)},$$

und erhalten

$$\log P(K|S_1, \dots, S_n) = \log P(S_1|K) + \cdots + \log P(S_n|K) + \log P(K) \\ \text{ } - \log P(S_1, \dots, S_n). \quad (11.18)$$

Um einen Score zu erhalten, müssen wir die Variablen $S_1, \dots, S_n$ als numerische Variablen mit den Werten 1 und 0 interpretieren. Das können wir leicht erkennen

$$\log P(S_i|K) = (\log P(S_i = 1|K) - \log P(S_i = 0|K))S_i + \log P(S_i = 0|K).$$

Es folgt dem

$$\log P(S_i|K) = \sum_{i=1}^N (\log P(S_i = 1|K) - \log P(S_i = 0|K))S_i \\ + \sum_{i=1}^N \log P(S_i = 0|K).$$

Nun definieren wir $w_i = \log P(S_i = 1|K) - \log P(S_i = 0|K)$ und $c = \sum_{i=1}^N \log P(S_i = 0|K)$ und vereinfachen

$$\log P(S_i|K) = \sum_{i=1}^N w_i S_i + c.$$

Eingesetzt in (11.18) erhalten wir

$$\log P(K|S_1, \dots, S_n) = \sum_{i=1}^N w_i S_i + c + \log P(K) - \log P(S_1, \dots, S_n).$$

Für die Entscheidung K muss nach der Definition des Bayes-Klassifikators gelten, dass $\log P(K|S_1, \dots, S_n) > \log(1/2)$. Es muss also entweder so sein

$$\sum_{i=1}^N w_i S_i + c + \log P(K) - \log P(S_1, \dots, S_n) > \log(1/2)$$

oder das

$$\sum_{i=1}^N w_i S_i > \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n),$$

womit wir einen Score mit dem Schwellenwert $\tilde{y} = \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n)$ definiert haben. Da alle Transformationen rückgängig gemacht werden können, können wir auch jeden beliebigen Score in einen Bayes'schen Klassifikator umwandeln. Damit ist die Äquivalenz nachgewiesen.

Übung 8.17 Die Logarithmierung von (8.10) auf Seite 206 ergibt

$$\log P(I | s_1, \dots, s_n) = \log c + \log P(I) + \sum_{i=1}^n \log P(w_i | I).$$

Dadurch werden aus sehr kleinen negativen Zahlen moderate negative Zahlen. Da die Logarithmusfunktion monoton wächst, maximieren wir zur Bestimmung der Klasse gemäß der Regel

$$I_{\text{Naive-Bayes}} = \arg \max_{I \in \{w, f\}} \log P(I | s_1, \dots, s_n).$$

Der Nachteil dieser Methode ist die etwas längere Rechenzeit in der Lernphase bei großen Texten. Beim Klassifizieren erhöht sich die Zeit nicht, da die Werte $\log P(I | s_1, \dots, s_n)$ beim Lernen gespeichert werden können.

Aufgabe 8.19 Sei f streng monoton wachsend, also $\tilde{y}x, yx < y \tilde{y} f(x) < f(y)$. Wenn nun $d_1(s, t) < d_1(u, v)$, dann ist eindeutig $d_2(s, t) = f(d_1(s, t)) < f(d_1(u, v)) = d_2(u, v)$. Da auch die Umkehrung von f streng monoton ist, gilt das Gegenteil, d. h. $d_2(s, t) < d_2(u, v) \tilde{y} d_1(s, t) < d_1(u, v)$. Somit wurde gezeigt, dass $d_2(s, t) < d_2(u, v) \tilde{y} d_1(s, t) < d_1(u, v)$.

Übung 8.20

$$\begin{aligned} 23 \cdot 26 \times 1 \times 2 &= 4, \text{ und somit } ds(x_1, x_2) = \sqrt[2]{\frac{23 \cdot 26}{4}} = 6,114 \tilde{y} 26 \cdot 20 \\ &= 2, \text{ und somit } ds(x_2, x_3) = \sqrt[2]{\frac{20 \cdot 23}{4}} = 11,4 \\ 23 \cdot 20 \times 1 \times 3 &= 1, \text{ und somit } ds(x_1, x_3) = \sqrt[2]{\frac{23 \cdot 20}{1}} = 21,41 \end{aligned}$$

Die Sätze 1 und 2 sind hinsichtlich der Distanzmetrik ds am ähnlichsten.

Übung 8.21 Hilfe bei Problemen mit KNIME: www.knime.org/forum

11.9 Neuronale Netze

Aufgabe 9.1 Wir wollen zeigen, dass $f(\tilde{y} + x) + f(\tilde{y} - x) = 1$.

$$f(\tilde{y} + x) = 1 + \frac{1}{e^{\frac{x}{T_e}}} = \frac{T_e}{1 + e^{\frac{x}{T_e}}}, \quad f(\tilde{y} - x) = 1 + \frac{1}{e^{-\frac{x}{T_e}}} = \frac{T_e}{1 + e^{-\frac{x}{T_e}}},$$

$$f(\tilde{y} + x) + f(\tilde{y} - x) = 1.$$

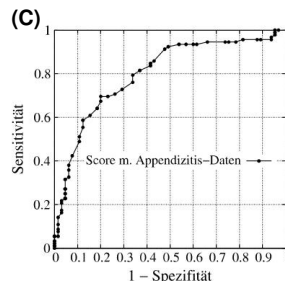
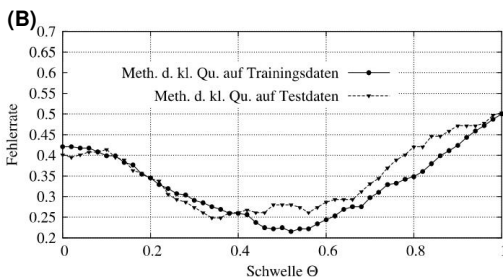
Aufgabe 9.3 Jedes im Netzwerk gespeicherte Muster hat eine Größe von n Bits. Das Netzwerk hat insgesamt $n(n+1)/2$ Gewichte. Wenn wir 16 Bits pro Gewicht reservieren und den Binärspeicher der Größe $16n(n+1)/2$ als gleich groß definieren, dann können damit eindeutig $N = 8n(n+1)/n = 4(n+1)$ Muster n Bits gespeichert werden in Größe. Für große n erhalten wir $N \approx 4n$. Bildet man wie in (9.11) auf Seite 237 den Quotienten $\tilde{y}$ aus der Anzahl der speicherbaren Bits und der Anzahl der verfügbaren Speicherzellen, so erhält man für den Listenspeicher 2 den Wert 1 und den Wert $\tilde{y} = 1/(4(n+1)/2) \approx 0,018$ für das Hopfield-Netzwerk. Der klassische Speicher hat somit (bei 16 Bit pro Gewicht) eine etwa 55-mal höhere Kapazität.

Übung 9.4

(a) Mathematica-Programm für die Methode der kleinsten Quadrate.

```
LeastSq[q_,a_] := Module[{Nq,Na,m,A,b,w}, Nq = Länge[q]; m
= Länge[q[[1]]]; Na = Länge[a]; If[Nq != Na, Print["Length[q] != Länge[a]"];
Ausgang, 0]; A = Tabelle[N[Summe[q[[p,i]] q[[p,j]], {p,1,Nq}]], {i,1,m}, {j,1,m}]; b =
Table[N[Sum[a[[p]] q[[p,i]], {p,1,Nq}]], {i,1,m}]; w = LinearSolve[A,b]
]
```

LeastSq::usage = "LeastSq[x,y,f] berechnet aus den Abfragevektoren q[[1]],..., q[[m]] eine Tabelle mit Koeffizienten w[[i]] für eine lineare Abbildung f[x] = Summe[w[[i]] x[[i]], {i,1,m}] mit f[q[[p]]] = a[[p]]."



Aufgabe 9.6 (a) Lernen funktioniert fehlerfrei. (b) Lernen funktioniert nicht ohne Fehler!

Aufgabe 9.7

- (a) Eine Abbildung f heißt linear, wenn für alle x, y, k gilt: $f(x + y) = f(x) + f(y)$ und $f(kx) = kf(x)$. Seien nun f und g lineare Abbildungen.
 Dann ist $f(g(x + y)) = f(g(x) + g(y)) = f(g(x)) + f(g(y))$ und $f(g(kx)) = f(kg(x)) = kf(g(x))$.
 Somit handelt es sich bei aufeinanderfolgenden Ausführungen linearer Abbildungen um eine lineare Abbildung.
- (b) Wir beobachten zwei beliebige Ausgabeneuronen j und k . Jeder der beiden repräsentiert eine Klasse. Die Klassifizierung erfolgt durch Bildung des Maximums der beiden Aktivierungen. $i w_{ji}$ und $net_j = i w_{ji}$ gewichtete Summe der Werte ar . Es sei $net_j = \text{riving}$ an den Neuronen j und k . Sei außerdem $net_j > net_k$. Ohne Aktivierungsfunktion wird Klasse j ausgegeben. Wenn nun eine streng monotone Aktivierungsfunktion f angewendet wird, ändert sich am Ergebnis nichts, da aufgrund der streng monotonen Funktion $f(net_j) > f(net_k)$ gilt.

Aufgabe 9.8 Der transformierte $f_2(x_1, x_2) = x_1^2 + x_2^2$. Dann ist die Trennlinie in der Raum $f_1(x_1, x_2) = x_1$ hat die Gleichung $y_1 + y_2 = 1$.

11.10 Verstärkungslernen

Aufgabe 10.1

- (a) n^N (b) $(n \cdot \bar{y} - 1)^N$ (C) 
- (D) 

Aufgabe 10.2 Werteiteration mit zeilenweiser Aktualisierung der V -Werte von links oben nach rechts unten ergibt

0	0	$\bar{y}$	0,81 0,9	$\bar{y}$	1,35 1,49	$\bar{y} \dots \bar{y}$	2,36 2,62
0	0		0,73 1		1,21 1,66		2,12 2,91

Übung 10.3

(C)

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	1	-1	1	-1	1	-1
5	1	-1	1	-1	1	-1
6	1	-1	1	-1	1	-1

6×6 feedback for a completely smooth surface

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	-1	1	-1	1	-1
3	1	-1	1	-1	1	-1
4	1	-1	1	-1	1	-1
5	1	-1	1	-1	1	-1
6	2	-2	2	-2	2	-2

6×6 feedback for a very heavy surface (like thick snow)

(d) Wir sehen, dass der Wert $\tilde{y}$ umso näher an 1 liegen muss, je länger eine Richtlinie wird (d. h. je mehr Schritte beispielsweise ein Zyklus einer zyklischen Strategie hat), da ein höherer Wert für $\tilde{y}$ zu einem längeren Gedächtnis führt möglich. Die Werteiteration konvergiert jedoch viel langsamer.

Aufgabe 10.4 Der Wert V folgt: $\tilde{y}(3, 3)$ unten rechts in der Zustandsmatrix wird geändert als

$$V^{\tilde{y}}(3, 1) = 0,9 V^{\tilde{y}}(2, 1) = 0,9 2V^{\tilde{y}}(2, 2) = 0,9 3V^{\tilde{y}}(2, 3) = 0,9 4V^{\tilde{y}}(3, 3). \tag{11.19}$$

Diese Gleichungskette folgt aus (10.6), weil für alle gegebenen Zustandsübergänge die maximale unmittelbare Belohnung $r(s, a) = 0$ ist, und das ist auch der Fall

$$V^{\tilde{y}}(s) = \max_A [r(s, a) + \tilde{y} V^{\tilde{y}}(\tilde{y}(s, a))] = \tilde{y} V^{\tilde{y}}(\tilde{y}(s, a)) = 0,9 V^{\tilde{y}}(\tilde{y}(s, a)).$$

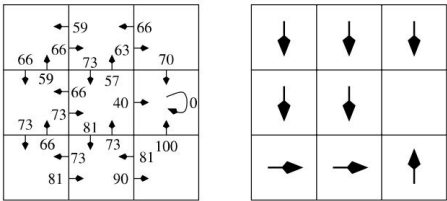
Aus (10.6) folgt auch, dass V maximal $\tilde{y}(3, 2) = 1 + 0,9 V^{\tilde{y}}(3, 1)$, weil $r(s, a) = 1$ ist. Analog gilt, dass $V^{\tilde{y}}(3, 3) = 1 + 0,9 V^{\tilde{y}}(3, 2)$ und der Kreis schließt. Die beiden letzten Gleichungen zusammen $\tilde{y}(3, 3) = 1 + 0,9(1 + 0,9 V^{\tilde{y}}(3, 1))$. Aus (11.19) folgt, dass V folgt $\tilde{y}$ ergeben $V(3, 1) = 0,9 4V^{\tilde{y}}(3, 3)$. Ersetzt in $V^{\tilde{y}}(3, 3)$, dies

$$V^{\tilde{y}}(3, 3) = 1 + 0,9(1 + 0,9 5V^{\tilde{y}}(3, 3)),$$

woraus sich die Behauptung ergibt.

Übung 10.5

Stabile Q-Werte und eine optimale Politik:



Aufgabe 10.6

(a) Anzahl der Zustände = $\tilde{y}^N$. Anzahl der Aktionen pro Zustand = $\tilde{y}^N$ $\tilde{y}^N$ 1. Anzahl der Policies = $(\tilde{y}^N)^{\tilde{y}^N}$ $n^{\tilde{y}^N}$.

(B)

	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 10$
$n = 1$	1	4	27	256	1010
$n = 2$	1	256	$3,9 \times 108$	$1,8 \times 1019$	10200
$n = 3$	$1,7 \times 107$	$n = 4$	$1,8 \times 4,4 \times 1038$	$3,9 \times 10115$	103000
1019	$3,9 \times 10154$	$n = 8$	$3,2 \times 10616$	$1,4 \times 3,2 \times 10616$	1040000
1025043	$6,7 \times 10315652$				10800000000

(c) Pro Staat gibt es nun 2^n mögliche Aktionen. Es gibt also $(2^n)^n$

ⁿ Richtlinien.

	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 10$
$n = 1$	2	4	8	16	1024
$n = 2$	4	16	64	256	1048576
$n = 3$	8	64	512	4096	1073741824
$n = 4$	16	256	2048	16384	109955149824
$n = 5$	32	1024	8192	65536	1125895364640
$n = 6$	64	4096	32768	262144	11529215043904
$n = 7$	128	16384	131072	1048576	118063617792000
$n = 8$	256	65536	524288	4194304	1208640000000000
$n = 9$	512	262144	2097152	16777216	12369431040000000
$n = 10$	1024	1048576	8388608	67108864	126554444800000000

(d) 10120411998 verschiedene Richtlinien können niemals kombinatorisch untersucht werden, selbst wenn alle der verfügbaren Computer der Welt sollten parallel darauf laufen.

Daher sind „intelligente“ Algorithmen erforderlich, um ein optimales oder nahezu optimales Ergebnis zu finden Politik.

Verweise

- [Ada75] EW Adams. *Die Logik der Bedingungen*. Synthese Library, Band 86. Reidel, Dordrecht, 1975.
- [Alp04] E. Alpaydin. *Einführung in maschinelles Lernen*. MIT Press, Cambridge, 2004.
- [APR90] J. Anderson, A. Pelionisz und E. Rosenfeld. *Neurocomputing (Band 2): Wegbeschreibung für die Forschung*. MIT Press, Cambridge, 1990.
- [AR88] J. Anderson und E. Rosenfeld. *Neurocomputing: Grundlagen der Forschung*. MIT Press, Cambridge, 1988. Sammlung grundlegender Originalarbeiten.
- [Bar98] R. Bartak. Online-Leitfaden zur Constraint-Programmierung, 1998. <http://kti.ms.mff.cuni.cz/~bartak/einschränkungen>.
- [BCDS08] A. Billard, S. Calinon, R. Dillmann und S. Schaal. Roboterprogrammierung durch Demonstration. In B. Siciliano und O. Khatib, Herausgeber, *Handbook of Robotics*, Seiten 1371–1394. Springer, Berlin, 2008.
- [Bel57] RE Bellman. *Dynamische Programmierung*. Princeton University Press, Princeton, 1957.
- [Ber89] M. Berrondo. Fallgruben für Kopffüssler. Fischer Taschenbuch, Nr. 8703, 1989.
- [BFOS84] L. Breiman, J. Friedman, RA Olshen und CJ Stone. *Klassifikations- und Regressionsbäume*. Wadsworth, Belmont, 1984.
- [Bib82] W. Bibel. *Automatisierte Beweisführung von Theoremen*. Vieweg, Wiesbaden, 1982.
- [Bis05] CM Bischof. *Neuronale Netze zur Mustererkennung*. Oxford University Press, London, 2005.
- [Bis06] CM Bischof. *Mustererkennung und maschinelles Lernen*. Springer, New York, 2006.
- [BM03] AG Barto und S. Mahadevan. Jüngste Fortschritte beim hierarchischen Verstärkungslernen. *Discrete Event Syst., Sonderausgabe zum Reinforcement Learning*, 13:41–77, 2003.
- [Bra84] V. Braitenberg. *Fahrzeuge – Experimente in der Synthetischen Psychologie*. MIT Press, Cambridge, 1984.
- [Bra01] B. Brabec. Computergestützte regionale Lawinenprognose. Doktorarbeit, ETH Zürich, 2001.
- [Bra11] I. Bratko. *PROLOG-Programmierung für künstliche Intelligenz*. Addison-Wesley, Reading, 4. Auflage, 2011.
- [Bri91] *Enzyklopädie Britannica*. Encyclopedia Britannica, London, 1991.
- [Bur98] CJ Burges. Ein Tutorial zu Support-Vektor-Maschinen zur Mustererkennung. *Daten Mindest. Wissen. Discov.*, 2(2):121–167, 1998.
- [CAD] CADE: Konferenz zum Thema Automatisierte Deduktion. www.cadeconference.org.
- [Che83] P. Cheeseman. Eine Methode zur Berechnung verallgemeinerter Bayes'scher Wahrscheinlichkeitswerte für Expertensysteme. In *Proc. des 8. Intl. Gemeinsame Konferenz. über künstliche Intelligenz (IJCAI 83)*, 1983.
- [Che85] P. Cheeseman. Zur Verteidigung der Wahrscheinlichkeit. In *Proc. des 9. Intl. Gemeinsame Konferenz. An Künstliche Intelligenz (IJCAI-85)*, 1985.
- [CL73] CL Chang und RC Lee. *Beweisen symbolischer Logik und mechanischer Theoreme*. Academic Press, Orlando, 1973.

- [Cle79] WS Cleveland. Robuste lokal gewichtete Regression und glättende Streudiagramme. *Marmelade. Stat. Assoc.*, 74(368):829–836, 1979.
- [CLR90] T. Cormen, Ch. Leiserson und R. Rivest. *Einführung in Algorithmen*. MIT Press, Cambridge, 1990.
- [CM94] WF Clocksin und CS Mellish. *Programmieren in Prolog*. Springer, Berlin, 4. Auflage, 1994.
- [Coz98] FG Cozman. Javabayes, Bayesianische Netzwerke in Java, 1998. www.cs.cmu.edu/~javabayes.
- [dD91] FT de Dombal. *Diagnose akuter Bauchschmerzen*. Churchill Livingstone, London, 1991.
- [dDLS+72] FT de Dombal, DJ Leaper, JR Staniland, AP McCann und JC Horrocks. Computergestützte Diagnose akuter Bauchschmerzen. *Br. Med. J.*, 2:9–13, 1972.
- [Dee11] Das DeepQA-Projekt, 2011. <http://www.research.ibm.com/deepqa/deepqa.shtml>.
- [DHS01] RO Duda, PE Hart und DG Stork. *Musterklassifizierung*. Wiley, New York, 2001.
- [Dia04] D. Diaz. GNU-PROLOG. Universität Paris, 2004. Aufl. 1.7, für GNU Prolog-Version 1.2.18, <http://gnu-prolog.inria.fr>.
- [DNM98] CL Blake, DJ Newman, S. Hettich und CJ Merz. UCI-Repository der Maschine Lerndatenbanken, 1998. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [Ede91] E. Eder. *Relative Komplexität von Kalkülen erster Ordnung*. Vieweg, Wiesbaden, 1991.
- [Elk93] C. Elkan. Der paradoxe Erfolg der Fuzzy-Logik. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, Seiten 698–703. MIT Press, Cambridge, 1993.
- [Ert93] W. Ertel. *Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen*. DISKI, Band 25. Infix, St. Augustin, 1993. Dissertation, Technische Universität München.
- [Ert11] W. Ertel. *Künstliche Intelligenz*, 2011. www.hs-weingarten.de/~ertel/aibook. Homepage zu diesem Buch mit Materialien, Demoprogrammen, Links, Literatur, Errata usw.
- [ES99] W. Ertel und M. Schramm. Kombination von Daten und Wissen durch MaxEnt-Optimierung von Wahrscheinlichkeitsverteilungen. In *PKDD'99 (3. Europäische Grundsatzkonferenz und Praxis der Wissensentdeckung in Datenbanken)*. LNCS, Band 1704, Seiten 323–328. Springer, Prag, 1999.
- [ESCT09] W. Ertel, M. Schneider, R. Cubek und M. Tokic. Der Lehrkasten: ein Universelles Roboter-Lernrahmen. In *Proceedings der 14. Internationalen Konferenz on Advanced Robotics (ICAR 2009)*, 2009. www.servicerobotik.hs-weingarten.de/Lehrbox.
- [ESS89] W. Ertel, J. Schumann und Ch. Suttner. Lernheuristik für einen Theorembeweis unter Verwendung der Rückausbreitung. In J. Retti und K. Leidlmair, Herausgeber, *5. Österreichische Artificial-Intelligence-Tagung*. Informatik-Fachberichte, Band 208, Seiten 87–95. Springer, Berlin, 1989.
- [Fit96] M. Passend. *Logik erster Ordnung und automatisierter Theorembeweis*. Springer, Berlin, 1996.
- [FNA+09] D. Ferrucci, E. Nyberg, J. Allan, K. Barker, E. Brown, J. Chu-Carroll, A. Ciccolo, P. Duboue, J. Fan, D. Gondek et al. Zur offenen Weiterentwicklung der Frage Antwortsysteme. IBM Technischer Bericht RC24789, Yorktown Heights, NY, 2009. http://www.research.ibm.com/deepqa/question_answering.shtml.
- [FPP07] D. Freedman, R. Pisani und R. Purves. *Statistiken*. Norton, New York, 4. Auflage, 2007.
- [Fra05] C. Frayn. Theorie der Computerschachprogrammierung, 2005. www.frayn.net/beowulf/theory.html.
- [Fre97] E. Freuder. Auf der Suche nach dem Heiligen Gral. *Einschränkungen*, 2(1):57–61, 1997.
- [FS97] B. Fischer und J. Schumann. Setheo geht in die Softwareentwicklung: Anwendung von ATP zur Wiederverwendung von Software. In *der Konferenz zum automatisierten Abzug (CADE-14)*. LNCS, Band 1249, Seiten 65–68. Springer, Berlin, 1997. <http://ase.arc.nasa.gov/people/schumann/publications/papers/cade97-reuse.html>.

- [GW08] RC González und RE Woods. *Digitale Bildverarbeitung*. Pearson/Prentice Hall, Upper Saddle River/Princeton, 2008.
- [Göd31a] K. Gödel. Diskussion zur Grundlegung der Mathematik, Erkenntnis 2. *Monatshefte Mathematik. Phys.*, 32(1):147–148, 1931.
- [Göd31b] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und veränderter Systeme I. *Monatshefte Math. Phys.*, 38(1):173–198, 1931.
- [HKP91] J. Hertz, A. Krogh und R. Palmer. *Einführung in die Theorie der neuronalen Berechnung*. Addison-Wesley, Reading, 1991.
- [Hon94] B. Hontschik. *Theorie und Praxis der Appendektomie*. Mabuse, Frankfurt am Main, 1994.
- [Hop82] JJ Hopfield. Neuronale Netze und physikalische Systeme mit entstehenden kollektiven Rechenfähigkeiten. *Proz. Natl. Acad. Wissenschaft. USA*, 79:2554–2558, 1982. Nachdruck in [AR88], Seiten 460–464.
- [HT85] JJ Hopfield und DW Tank. „Neuronale“ Berechnung von Entscheidungen in der Optimierung Probleme. *Biol. Cybern.*, 52(3):141–152, 1985. Springer.
- [HTF09] T. Hastie, R. Tibshirani und J. Friedman. *Die Elemente des statistischen Lernens: Daten Bergbau, Schlussfolgerung und Vorhersage*. Springer, Berlin, 3. Auflage, 2009. Online-Version: <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>.
- [Jay57] ET Jaynes. Informationstheorie und statistische Mechanik. *Physik. Rev.*, 1957.
- [Jay03] ET Jaynes. *Wahrscheinlichkeitstheorie: Die Logik der Wissenschaft*. Cambridge University Press, Cambridge, 2003.
- [Jen01] FV Jensen. *Bayesianische Netzwerke und Entscheidungsgraphen*. Springer, Berlin, 2001.
- [Jor99] MI Jordan, Herausgeber. *Lernen in grafischen Modellen*. MIT Press, Cambridge, 1999.
- [Kal01] JA Kalman. *Automatisiertes Denken mit OTTER*. Rinton, Paramus, 2001. www-unix.mcs.anl.gov/AR/otter/index.html.
- [Kan89] Th. Kane. Maximale Entropie in Nilssons probabilistischer Logik. In *Proc. vom 11 International Gemeinsame Konferenz. über künstliche Intelligenz (IJCAI-89)*, 1989.
- [KK92] JN Kapur und HK Kesavan. *Prinzipien der Entropieoptimierung mit Anwendungen*. Academic Press, San Diego, 1992.
- [KLM96] LP Kaelbling, ML Littman und AP Moore. Reinforcement Learning: eine Umfrage. *J. Artif. Intel. Res.*, 4:237–285, 1996. www-2.cs.cmu.edu/afs/cs/project/jair/pub/Band4/kaelbling96a.pdf.
- [KMK97] H. Kimura, K. Miyazaki und S. Kobayashi. Verstärkungslernen in POMDPs mit Funktionsnäherung. In *14. International Conference on Machine Learning*, Seiten 152–160. Morgan Kaufmann, San Mateo, 1997. <http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robdemo.html>.
- [Koh72] T. Kohonen. Korrelationsmatrixspeicher. *IEEE Trans. Comput.*, C-21(4):353–359, 1972. Nachdruck in [AR88], Seiten 171–174.
- [Lar00] FD Laramée. Schachprogrammierung, Teile 1–6, 2000. www.gamedev.net/reference/Programmierung/Funktionen/Schach1.
- [Le999] LEXMED – ein lernendes Expertensystem für die medizinische Diagnose, 1999. www.lexmed.de.
- [Lov78] DW Loveland. *Automatisierte Theoremprüfung: Eine logische Grundlage*. Nordholland, Amsterdam, 1978.
- [LSBB92] R. Letz, J. Schumann, S. Bayerl und W. Bibel. SETHEO: ein leistungsstarker Theorembeweiser. *J. Autom. Reason.*, 8(2):183–212, 1992. www.informatik.tu-muenchen.de/~letz/setheo.
- [McC] W. McCune. Automatisierte Abzugssysteme und -gruppen. www-unix.mcs.anl.gov/AR/andere.html. Siehe auch www-formal.stanford.edu/ct/ARS/systems.html.
- [McD82] J. McDermott. R1: ein regelbasierter Konfigurator von Computersystemen. *Artif. Intellektuell*, 19:39–88, 1982.
- [MDBM00] G. Melancon, I. Dutour und G. Bousque-Melou. Zufällige Generierung von Dags für Diagrammzeichnung. Technischer Bericht INS-R0005, Niederländisches Forschungszentrum für Mathematik und Informatik (CWI), 2000. <http://ftp.cwi.nl/CWIreports/INS/INS-R0005.pdf>.

- [Mit97] T. Mitchell. *Maschinelles Lernen*. McGraw-Hill, New York, 1997. www-2.cs.cmu.edu/~tom/mlbook.html.
- [MP69] M. Minsky und S. Papert. *Perceptrone*. MIT Press, Cambridge, 1969.
- [Neu00] M. Neugeborenes. *Automatisierte Beweisführung von Theoremen: Theorie und Praxis*. Springer, Berlin, 2000.
- [Nil86] N.J. Nilsson. Wahrscheinlichkeitslogik. *Artif. Intell.*, 28(1):71–87, 1986.
- [Nil98] N. Nilsson. *Künstliche Intelligenz – eine neue Synthese*. Morgan Kaufmann, San Mateo, 1998.
- [NPW02] T. Nipkow, L.C. Paulson und M. Wenzel. *Isabelle/HOL – Ein Beweisassistent für Logik höherer Ordnung*. LNCS, Band 2283. Springer, Berlin, 2002. www.cl.cam.ac.uk/Research/HVG/Isabelle.
- [NS61] A. Newell und H.A. Simon. GPS, ein Programm, das menschliches Denken simuliert. In H. Billing, Herausgeber, *Lernende Automaten*, Seiten 109–124. Oldenburg, München, 1961.
- [NSS83] A. Newell, J.C. Shaw und H.A. Simon. Empirische Untersuchungen mit der Logik Theoriemaschine: eine Fallstudie zur Heuristik. In J. Siekmann und G. Wrightson, Herausgeber, *Automatisierung des Denkens 1: Klassische Arbeiten zur Computerlogik 1957–1966*, Seiten 49–73. Springer, Berlin, 1983. Erstveröffentlichung: 1957.
- [OFY+95] C. Ohmann, C. Franke, Q. Yang, M. Margulies, M. Chan, van P.J. Elk, F.T. de Dombal und H.D. Röher. Diagnosescore für akute Appendizitis. *Chirurg*, 66:135–141, 1995.
- [OMYL96] C. Ohmann, V. Moustakis, Q. Yang und K. Lang. Evaluierung automatischer Wissenserwerbstechiken bei der Diagnose akuter Bauchschmerzen. *Artif. Intel. Med.*, 8:23–36, 1996.
- [OPB94] C. Ohmann, C. Platen und G. Belenky. Computerunterstützte Diagnose bei akuten Bauchschmerzen. *Chirurg*, 63:113–123, 1994.
- [Pal80] G. Palm. Zum assoziativen Gedächtnis. *Biol. Cybern.*, 36:19–31, 1980.
- [Pal91] G. Palm. Speicherkapazitäten lokaler Regeln für synaptische Modifikation. *Concepts Neu rosci.*, 2(1):97–128, 1991. MPI Tübingen.
- [Pea84] J. Perle. *Heuristiken, intelligente Suchstrategien zur Lösung von Computerproblemen*. Addison-Wesley, Reading, 1984.
- [Pea88] J. Perle. *Probabilistisches Denken in intelligenten Systemen. Netzwerke plausibler Schlussfolgerungen*. Morgan Kaufmann, San Mateo, 1988.
- [PL05] L. Panait und S. Luke. Kooperatives Lernen mit mehreren Agenten: der Stand der Technik. *Auton. Agents Multi-Agent Syst.*, 11(3):387–434, 2005.
- [PS08] J. Peters und S. Schaal. Verstärkendes Erlernen motorischer Fähigkeiten mit Politikgradienten. *Neural Netw.*, 21(4):682–697, 2008. <http://www-clmc.usc.edu/publications/P/peters-NN2008.pdf>.
- [PS09] G. Pólya und S. Sloan. *Wie man es löst: Ein neuer Aspekt der mathematischen Methode*. Ishi, Mountain View, 2009.
- [Qui93] J. Ross Quinlan. *C4.5: Programme für maschinelles Lernen*. Morgan Kaufmann, San Mateo, 1993. C4.5-Download: <http://www.rulequest.com/Personal>, C5.0-Download: <http://www.rulequest.com/download.html>.
- [RB93] M. Riedmiller und H. Braun. Eine direkte adaptive Methode für eine schnellere Backpropagation Lernen: der Rprop-Algorithmus. In *Proceedings der IEEE International Conference on Neural Networks*, Seiten 586–591, 1993.
- [RGH+06] M. Riedmiller, T. Gabel, R. Hafner, S. Lange und M. Lauer. Die Brainstormer: Entwurfsprinzipien lernfähiger autonomer Roboter. *Inform.-Spektrum*, 29(3):175–190, 2006.
- [RHR86] D.E. Rumelhart, G.E. Hinton und R.J. Williams. Interne Repräsentationen lernen durch Fehlerfortpflanzung. In [RM86].
- [Ric83] E. Rich. *Künstliche Intelligenz*. McGraw-Hill, New York, 1983.
- [RM86] D. Rumelhart und J. McClelland. *Parallele verteilte Verarbeitung*, Band 1. MIT Press, Cambridge, 1986.

- [RM96] W. Rödder und C.-H. Meyer. Kohärente Wissensverarbeitung bei maximaler Entropie von SPIRIT. In KI-96 (*Bundeskonferenz KI*), Dresden, 1996.
- [RMD07] M. Riedmiller, M. Montemerlo und H. Dahlkamp. Lernen, ein echtes Auto zu fahren 20 Minuten. In FBIT '07: *Proceedings of the 2007 Frontiers in the Convergence of Biowissenschaften und Informationstechnologien*, Seiten 645–650. IEEE Computer Society, Washington, D.C., 2007.
- [RMS92] H. Ritter, T. Martinez und K. Schulten. *Neuronale Berechnung und Selbstorganisation Karten*. Addison-Wesley, Reading, 1992.
- [RN10] S. Russell und P. Norvig. *Künstliche Intelligenz: Ein moderner Ansatz*. Prentice Hall, New York, 3. Auflage, 2010. 1. Auflage: 1995. <http://aima.cs.berkeley.edu>.
- [Roba] Offizielle Robocup-Website. www.robocup.org.
- [Robb] Der Robocup-Fußballsimulator. <http://sserver.sourceforge.net>.
- [Rob65] JA Robinson. Eine maschinenorientierte Logik basierend auf dem Auflösungsprinzip. *J. ACM*, 12(1):23–41, 1965.
- [Roj96] R. Rojas. *Neuronale Netze: Eine systematische Einführung*. Springer, Berlin, 1996.
- [Ros58] F. Rosenblatt. Das Perzeptron: ein probabilistisches Modell zur Informationsspeicherung und Organisation im Gehirn. *Psychol. Rev.*, 65:386–408, 1958. Nachdruck in [AR88], Seiten 92–114.
- [Ros09] SM Ross. *Einführung in die Wahrscheinlichkeitstheorie und Statistik für Ingenieure und Wissenschaftler*. Academic Press, San Diego, 2009.
- [RW06] CE Rasmussen und CKI Williams. *Gaußsche Prozesse für maschinelles Lernen* ing. MIT Press, Cambridge, 2006. Online-Version: <http://www.gaussianprocess.org/gpml/chapters/>.
- [SA94] S. Schaal und CG Atkeson. Roboter-Jonglage: speicherbasierte Implementierung Lernen. *IEEE-Steuerungssystem. Mag.*, 14(1):57–71, 1994.
- [Sam59] AL Samuel. Einige Studien zum maschinellen Lernen mithilfe des Damespiels. *IBM J.*, 1(3):210–229, 1959.
- [SB98] R. Sutton und A. Barto. *Verstärkungslernen*. MIT Press, Cambridge, 1998. www.cs.ualberta.ca/~sutton/book/the-book.html.
- [SB04] J. Siekmann und Ch. Benzmlüller. Omega: Computergestützte Mathematik. In KI 2004: *Fortschritte in der künstlichen Intelligenz*. LNAI, Band 3238, Seiten 3–28. Springer, Berlin, 2004. www.ags.uni-sb.de/~omega.
- [Sch96] M. Schramm. *Indifferenz, Unabhängigkeit und maximale Entropie: Eine Wahrscheinlichkeitstheoretische Semantik für Nicht-Monotones Schließen*. Dissertation zur Informatik, Band 4. CS, München, 1996.
- [Sch01] J. Schumann. *Automatisierte Beweisführung von Theoremen in der Softwareentwicklung*. Springer, Berlin, 2001.
- [Sch02] S. Schulz. E – ein intelligenter Theorembeweiser. *J. AI Commun.*, 15(2/3):111–126, 2002. www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html.
- [Sch04] A. Schwartz. *SpamAssassin*. O'Reilly, Cambridge, 2004. SpamAssassin-Homepage: <http://spamassassin.apache.org>.
- [SE90] CH. Suttner und W. Ertel. Automatische Erfassung suchführender Heuristiken. Im *10 Int. Konf. zum automatischen Abzug*. LNAI, Band 449, Seiten 470–484. Springer, Berlin, 1990.
- [SE00] M. Schramm und W. Ertel. Argumentation mit Wahrscheinlichkeiten und maximaler Entropie: das System PIT und seine Anwendung in LEXMED. In K. Inderfurth et al., Herausgeber, *Operations Research Proceedings (SOR'99)*, Seiten 274–280. Springer, Berlin, 2000.
- [SE10] M. Schneider und W. Ertel. Roboterlernen durch Demonstration mit lokalem Gaussian Prozessregression. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10)*, 2010.
- [SET09] T. Segaran, C. Evans und J. Taylor. *Programmierung des Semantic Web*. O'Reilly, Cambridge, 2009.
- [Sho76] EH Shortliffe. *Computergestützte medizinische Beratung, MYCIN*. Nordholland, New York, 1976.

- [Spe97] *Geheimnisse des Geistes*. Sonderausgabe. Scientific American Inc., 1997.
- [Spe98] *Intelligenz erforschen*. Scientific American Presents, Band 9. Scientific American Inc., 1998.
- [SR86] TJ Sejnowski und CR Rosenberg. NETtalk: ein paralleles Netzwerk, das lesen lernt laut. Technischer Bericht JHU/EECS-86/01. The John Hopkins University Electrical Engineering and Computer Science Technical Report, 1986. Nachdruck in [AR88], Seiten 661–672.
- [SS02] S. Schölkopf und A. Smola. *Lernen mit Kernen: Support Vector Machines, Regularisierung, Optimierung und mehr*. MIT Press, Cambridge, 2002.
- [SS06] G. Sutcliffe und C. Sutton. Der Stand von CASC. *AI Commun.*, 19(1):35–48, 2006. CASC-Homepage: www.cs.miami.edu/~tptp/CASC.
- [SSK05] P. Stone, RS Sutton und G. Kuhlmann. Verstärkungslernen für Robocup-Fußball-Keepaway. *Adaptives Verhalten*, 2005. www.cs.utexas.edu/~pstone/Papers/bib2html-links/AB05.pdf.
- [Ste07] J. Stewart. *Multivariable Infinitesimalrechnung*. Brooks Cole, Florenz, 2007.
- [SW76] CE Shannon und W. Weaver. *Mathematische Grundlagen der Informationstheorie*. Oldenbourg, München, 1976.
- [Größe10] C. Szepesvári. *Algorithmen für Reinforcement Learning*. Morgan & Claypool, San Rafael, 2010. Entwurf online verfügbar: <http://www.ualberta.ca/~szepesva/RLBook.html>.
- [Ted08] R. Tedrake. Lernkontrolle bei mittleren Reynolds-Zahlen. Im *Workshop zu: Robotics Challenges for Machine Learning II, Internationale Konferenz über intelligente Roboter und Systeme (IROS 2008)*, Nizza, Frankreich, 2008.
- [TEF09] M. Tokic, W. Ertel und J. Fessler. Der Crawler, ein Klassenzimmer-Demonstrator für verstärkendes Lernen. In *Proceedings of the 22nd International Florida Artificial Konferenz der Intelligence Research Society (FLAIRS 09)*. AAAI Press, Menlo Park, 2009.
- [Tes95] G. Tesauro. Zeitdifferenzlernen und TD-Gammon. *Komm. ACM*, 38(3), 1995. www.research.ibm.com/massive/tdl.html.
- [Tok06] M. Tokic. Entwicklung eines lernfähigen Laufroboters. Diplomarbeit Hochschule Ravensburg-Weingarten, 2006. Inklusive Simulationssoftware verfügbar auf www.hs-weingarten.de/~ertel/kibuch.
- [Tur37] AM Turing. Über berechenbare Zahlen, mit einer Anwendung auf das Entscheidungsproblem. *Proz. London. Mathematik. Soc.*, 42(2), 1937.
- [Tur50] AM Turing. Computermaschinen und Intelligenz. *Mind*, 59:433–460, 1950.
- [vA06] L. v. Ahn. Spiele mit einem Zweck. *IEEE Comput. Mag.*, 96–98, Juni 2006. <http://www.cs.cmu.edu/~biglou/ieee-gwap.pdf>.
- [Wei66] J. Weizenbaum. ELIZA – ein Computerprogramm zur Untersuchung der natürlichsprachlichen Kommunikation zwischen Mensch und Maschine. *Komm. ACM*, 9(1):36–45, 1966.
- [WF01] I. Witten und E. Frank. *Data Mining*. Hanser, München, 2001. DataMining Java Library WEKA: www.cs.waikato.ac.nz/~ml/weka.
- [Whi96] J. Whittaker. *Grafische Modelle in der angewandten multivariaten Statistik*. Wiley, New York, 1996.
- [Wie] U. Wiedemann. PhilLex, Lexikon der Philosophie. www.phillex.de/paradoxa.htm.
- [Wie04] J. Wielemaker. SWI-PROLOG 5.4. Universität van Amsterdam, 2004. www.swi-prolog.org.
- [Wik10] Wikipedia, die freie Enzyklopädie, 2010. <http://en.wikipedia.org>.
- [Gewinnen] P. Winston. Spieldemonstration. <http://www.ai.mit.edu/courses/6.034f/gamepair.html>. Java-Applet für Minimax- und Alpha-Beta-Suche.
- [Zel94] A. Zell. *Simulation neuronaler Netze*. Addison-Wesley, Reading, 1994. Beschreibung von SNNS und JNNS: www-ra.informatik.uni-tuebingen.de/SNNS.
- [ZSR+99] A. Zieles, H. Sitter, TA Rampp, E. Schäfer, C. Hasse, W. Lorenz und M. Rothmund. Überprüfung eines diagnostischen Scoresystems (Ohmann-Score) für die akute Blinddarmentzündung. *Chirurg*, 70:777–783, 1999.

Index

A

A-priori-Wahrscheinlichkeit,
 A^y 122-Algorithmus, 98, 250
 Aktion, 86, 87, 260
 Aktivierungsfunktion, 224, 234
 Aktuator, 9
 Adaptive Resonanztheorie, 255
 Zulässig, 98
 Agent, 3, 9, 257, 259–261, 265, 267, 268, 270, 272, 273
 autonom,
 8 kostenbasiert,
 12 kostenorientiert,
 140 verteilt, 8
 zielbasiert, 9
 Hardware, 9
 intelligent, 9
 Lernen, 8, 12,
 164 Reflex, 9 Software,
 9 Utility-
 basiert, 12
 mit Gedächtnis, 9
 Agenten, verteilt, 12
 KI, 1
 Alarmbeispiel, 145
 Alpha-Beta-Schnitt, 103
 Und Filialen, 70
 Und-oder Baum, 70
 Blinddarmentzündung, 121, 131
 Annäherung, 164, 180
 KUNST, 255
 Künstliche Intelligenz, 1
 Assoziatives Gedächtnis, 232
 Attribut, 106, 185
 Autoassoziatives Gedächtnis, 226, 233, 238

B

Backgammon, 108, 273
 Backpropagation, 246, 265, 275
 Lernregel, 247

Backtracking, 69, 91
 Rückwärtsverkettung, 27
 Batch-Lernen, 201, 243
 Bayes-
 Formel, *siehe* Bayes-Theorem-
 Theorem, 122, 155
 Bayes-Formel, *siehe* Bayes-Theorem
 Bayesianisches Netzwerk, 7, 10, 64, 115, 144, 146,
 184
 Lernen, 157, 199
 BDD, *siehe* binäres Entscheidungsdiagramm
 Bellman-
 Gleichung, 263-
 Prinzip, 263
 Bias-Einheit, 173
 Binäres Entscheidungsdiagramm, 29
 Boltzmann-Maschine, 232
 Gehirnwissenschaft, 3
 Braitenberg-Fahrzeug, 1, 8, 179
 Verzweigungsfaktor: 83, 87
 Durchschnitt,
 84 effektiv, 87
 Eingebautes Prädikat, 76

 C C4.5, 184, 200
 Analysis, 20
 Gentzen, 40
 natürliches Denken, 40
 sequentielles
 Denken, 40 CART,
 184, 198
 CASC, 48
 Fallbasis, 183 Fallbasiertes
 Denken, 183 CBR, *siehe* fallbasiertes
 Denken,
 Sicherheitsfaktoren, 114 Kettenregel für Bayes'sche

 Netzwerke, 120,
 154, 155 Chatterbot, 5 Checkers, 102, 103, 108

- Schach, 102, 103, 105, 108
- Kirche, Alonso, 5
- Klassifizierung, 162
- Klassifikator, 162, 164, 245
- Klausel, 21
 - Kopf, 26
 - definitiv, 26
- Geschlossene Formel, 32
- CLP, *siehe* Constraint-Logik-Programmierung
- Cluster, 206
- Clustering, 206, 216
 - hierarchisch, 209
- CNF, 21
- Kognitionswissenschaft, 3
 - Ergänzung, 23
- Komplett, 20
- Computerdiagnose, 144
- Fazit, 26
- Bedingte Wahrscheinlichkeit,
 - Tabelle 119, *siehe* CPT
- Bedingt unabhängig, 146, 147, 155
- Konditionierung, 149, 155
- Verwirrungsmatrix, 11, 212
- Konjunktion, 16, 21
- Konjunktive Normalform, 21
- Konnektionismus, 7
- Konsequent, 23
- Konstant, 32
- Constraint-Logik-Programmierung, 78
- Problem der Einschränkungszufriedenheit, 78
- Korrelation, 137
 - Koeffizient, 168
 - Matrix, 216
- Kostenschätzungsfunktion, 96
- Kostenfunktion, 87, 98
- Kostenmatrix, 136, 139
- CPT, 146, 149, 156, 199, 202
- Kreditvergabe, 107, 260
- Kreuzvalidierung, 198
- Übersprechen, 236
- CSP, 78
- Fluch der Dimensionalität, 274
- Schnitt, 71
- D**
- D-Trennung, 156
- DAG, 155, 200
- DAI, 8
- Data Mining, 8, 165, 166, 182, 184, 197
- De Morgan, 37
- Entscheidung, 139
- Entscheidungsbaum, 13,
 - 185 Induktion, 165, 185
 - Lernen, 184, 275
- Standardlogik, 63
- Standardregel, 63
- Delta-Regel, 243–245
 - verallgemeinert, 247
- Demodulation, 46
- Dempster-Schäfer-Theorie, 115
- Abhängigkeitsdiagramm, 137
- Tiefenbegrenzung: 91
- Ableitung, 20
- Deterministisch, 89, 102
- Diagnosesystem, 132
- Disjunktion, 16, 21
- Entfernungsmetrik, 207
- Verteilte künstliche Intelligenz, 8
- Verteiltes Lernen, 274
- Verteilung, 118, 134
- Dynamische Programmierung, 263
- E**
- E-Learning, 5
- Eifrig lernen, 182, 215
- Grundveranstaltung, 115
- Eliza, 5
- EM-Algorithmus, 201, 208
- Entropie,
 - maximal 188, 115, 122
- Umgebung, 9, 12
 - kontinuierlich, 12
 - deterministisch, 12
 - diskret, 12
 - nichtdeterministisch, 12
 - beobachtbar, 12
 - teilweise beobachtbar, 12
- Gleichung, gerichtet, 47
- Gleichwertigkeit, 16
- Bewertungsfunktion, 103
- Veranstaltung, 115
- Expertensystem, 131, 144
- F**
- Tatsache, 26
- Faktorisierung, 23, 45
- Falsch negativ, 141
- Falsch positiv, 140
- Algorithmus für den am weitesten entfernten Nachbarn, 211
- Feature, 106, 162, 171, 185
- Funktionsraum, 163
- Finite-Domain-Constraint-Löser, 79
- Satz erster Ordnung, 32
- Vorwärtsverkettung, 27
- Rahmenproblem, 63
- Freie Variablen, 32

Funktionssymbol, 32
Fuzzy-Logik, 8, 13, 29, 64, 115

G

Gaußscher Prozess, 181, 215, 272
Allgemeiner Problemlöser, 10
Verallgemeinerung, 162
Genetische Programmierung, 75
Los, 102, 103, 108
Tor, 28
 Stapel, 28
 Zustand, 87
Gödel
 Kurt,
 5er Vollständigkeitssatz, 5, 41er
 Unvollständigkeitssatz, 5, 60
GPS, 10
Gefälleabstieg, 244
Gierige Suche, 96, 97, 201
Grundbegriff, 41

H

Halteproblem, 5
Hebb-Regel, 225, 234, 247
 binär, 236
Heuristik, 46, 57, 94
Heuristische Auswertungsfunktion, 95, 98
Hierarchisches Lernen, 274
Hopfield-Netzwerk, 226, 227, 237
Hornklause, 26, 73
Hugin, 150

-

ID3, 184
IDA*-Algorithmus, 100, 250
Sofortige Belohnung, 260
Implikation, 16
Inkrementeller Gradientenabstieg, 245
Inkrementelles Lernen, 243
Unabhängig, 119
 bedingt, 146, 147, 155
Gleichgültigkeit, 126
Indifferente Variablen, 126
Inferenzmaschine, 42
Inferenzmechanismus, 12
Informationsgehalt, 189
Informationsgewinn, 186, 189, 215
Eingabeauflösung, 46
Interpretation, 16, 33
Iterative Vertiefung, 92, 100

J

JavaBayes, 150

K

K Methode des nächsten Nachbarn, 178, 180
K-bedeutet, 208
Kernel, 181, 253
Kernel-Methode, 253 k
Methode des nächsten Nachbarn, 180
KNIME, 185, 211, 212
Wissen, 12 Basis,
 12, 18, 145
 konsistent, 23
Ingenieur, 9, 12
Technik, 12
Quellen, 12

L

Laplace-Annahme, 117
Laplace-Wahrscheinlichkeiten, 117
Wirtschaftsrecht, 197
Faules Lernen, 182, 215
Lernen, 161
 Batch, 243
 durch Demonstration, 275
 verteilt, 274
 hierarchisch, 274
 inkrementell, 201, 243
 Maschine, 136
 Multi-Agent, 274
 Verstärkung, 89, 107, 214
 halbüberwacht, 214
 überwacht, 161, 206, 238
 TD-, 273
Lernagent, 164
Lernphase, 164
Lernrate: 225, 244
Kleinste Quadrate, 143, 241, 242, 245
LEXMED, 115, 122, 131, 193
Begrenzte Ressourcen, 95
Lineare Näherung, 245
Linear trennbar, 169, 170
LIPPEN, 68
LISP, 6, 10
Wörtlich, 21
 Komplementär, 23
Lokal gewichtete lineare Regression, 183
Fuzzy-
 Logik, 29, 64, 115
 höherer Ordnung, 59,
 60 Probabilistik, 13, 29
Logiktheoretiker, 6, 10
Logisch gültig, 17

M

Maschinelles Lernen, 134, 161
 Manhattan-Entfernung: 100, 207
 Randverteilung, 121
 Marginalisierung, 120, 121, 123, 155
 Markov-Entscheidungsprozess, 9, 273
 nichtdeterministisch, 270
 teilweise beobachtbar, 261
 Markov-Entscheidungsprozesse, 261
 Materielle Implikation, 16, 115, 128
 MaxEnt, 115, 126, 129, 131, 134, 149, 156 Verteilung,
 126

MDP, 261, 263, 273

 deterministisch, 269
 nichtdeterministisch, 270

Auswendiglernen, 161

Gedächtnisbasiertes Lernen, 182, 183

MGU, 44

Minimaler Spannbaum, 210

Modell, 17

Modus ponens, 22, 30, 114, 125

Dynamik, 251

Monoton, 61

Lernen mit mehreren Agenten, 274

Multiagentensysteme, 10

MYCIN, 10, 114, 132

N

Naive Bayes, 143, 144, 157, 166, 175, 202, 204, 219,
 298 Klassifikator,

 202, 204

Naive Rückseite, 74

Navier-Stokes-Gleichung, 274

Klassifizierung des

 nächsten Nachbarn,
 176-Methode, 175

Algorithmus für den nächsten Nachbarn, 210

Verneinung, 16

Negation als Misserfolg, 73

Neuronales Netzwerk, 6, 10, 180, 181, 221

Wiederkehrende

 neuronale Netze, 226, 231

Neuroinformatik, 231

Neurowissenschaften, 3

Neurotransmitter, 223

Lärm, 177

Nichtmonotone Logik, 63, 130

Normalgleichungen, 242

Normalform

 Konjunktiv, 21

 Pränex, 37

Ö

Observable, 89, 103

Occams Rasiermesser, 197

Offline-Algorithmus, 89

Online-Algorithmus, 89

Ontologie, 53

Oder Filialen, 70

Orthonormal, 234

Othello, 102, 103

Überanpassung, 177, 197, 198, 201, 240, 243, 252, 273

EULE, 53

P

Paradox, 60

Paramodulation, 47

Teilweise beobachtbarer Markov-Entscheidungsprozess,
 261

Pinguinproblem, 77

Perceptron, 10, 169, 170, 224

Phasenübergang, 230

PIT, 129, 130, 149, 157

PL1, 13, 32

Planung, 76

Richtlinie,

 260 Gradientenmethode,
 273 optimal, 260

POMDP, 261, 273

Nachbedingung, 52

Voraussetzung, 52

Prädikatenlogik, 5

 erster Ordnung, 13, 32

Präferenzlernen, 166

Prämisse, 26

Wahrscheinlichkeitslogik, 13, 63 Argumentation, 7

Wahrscheinlichkeit, 115,
 117 Verteilung, 118

 Regeln, 137

Produktregel, 120

Programmüberprüfung, 51

PROLOG, 7, 10, 20, 28, 67

Beweissystem, 19

Satzvariablen, 15

Aussagenkalkül,

 13 Logik, 15

Beschneiden, 192, 198

Pseudoinverse, 236

Reine wörtliche Regel, 46, 55

Q

Q-Learning, 267, 275

 Konvergenz, 269

Quickprop, 252

R

Zufallsvariable, 116

Rapid Prototyping, 79

RDF, 53

Echtzeitentscheidung, 95

Echtzeitanforderung, 103

Betriebscharakteristik des Empfängers, 142

Verstärkung

Lernen, 257, 260

negativ, 260

positiv, 260

Auflösung, 6, 22

Infinitesimalrechnung,

10, 20 Regel,

22, 30 Allgemeines, 22, 43

SLD, 27

Resolvent, 22

Prämie

ermäßigt, 260

sofort, 260, 269

Risikomanagement, 141

RoboCup, 11, 273

Roboter, 9, 258

Geh-, 258

Robotik, 257

ROC-Kurve, 143, 214

RProp, 252

S

Probe, 185

Zufrieden, 17

Streudiagramm, 163

Punktzahl: 132, 143, 203, 219, 242

Suchalgorithmus,

86 vollständig,

87 optimal,

89 heuristisch,

84 Leerzeichen, 23, 27,

40, 46

Baum, 86 uninformiert, 84

Selbstorganisierende Karten, 255

Semantische Bäume, 28

Semantisches Web, 53

Semantik

deklarativ (PROLOG), 70

prozedural (PROLOG), 70, 73

Halbentscheidbar, PL1, 59

Halbüberwachtes Lernen, 214

Empfindlichkeit, 141, 148

Sensor, 9

Satz von Unterstützungsstrategien, 46, 55

Sigmoidfunktion, 225, 241, 246

Unterschrift, 15

Ähnlichkeit, 175

Simuliertes Glühen, 232

Situationsrechnung, 63

Skolemisierung, 39

SLD-Auflösung, 30

Software-Wiederverwendung, 52

Lösung, 87

Ton, 20

Spam, 204-

Filter, 11, 204

Spezifität, 141

Ausgangszustand, 87

Zustand, 87, 258, 259

Raum, 87

Übergangsfunktion, 260

Statistische Induktion, 136

Unterziel, 28, 69

Substitutionsaxiom, 36

Subsumtion, 46

Support-Vektor, 252

Maschine, 252, 272

Support-Vektor-Maschine, 181

SVM, *siehe* Support Vector Machine

T

Zielfunktion, 164

Tautologie, 17

TD

-error, 271

-gammon, 273

-learning, 271, 273

Lehrbox, 275

Zeitdifferenzfehler, 271

Lernen, 271

Begriff, 32

Umschreibung, 47

Testdaten, 164, 197

Text

Klassifizierung, 204

Bergbau, 166

Theorembeweiser, 6, 42, 43, 47, 51, 52

Trainingsdaten, 58, 164, 197

Übergangsfunktion, 270

Richtig, 34

Wahrheitstabelle,

16 Methode, 19

Turing

Alan, 5.

Test, 4

Tweety-Beispiel, 60, 63, 130

U

Unifizierbar, 44

Vereinigung, 44

Unifier, 44

am allgemeinsten, 44

Einheitliche Kostensuche, 90

Einheitsklausel,

46 Resolution, 46

Unerfüllbar, 17

V

Gültig, 17, 34

Wertiteration, 263

Variabel, 32

VDM-SL, 52

Spezifikation der Wiener Entwicklungsmethode

Sprache, 52

Voronoi-Diagramm, 177

W

Laufroboter, 258

WAM, 68, 75

Warren abstrakte Maschine, 68

Watson, 13

WEKA, 185, 211

símbolo de función, 32

Lógica difusa, 8, 13, 29, 64, 115

Proceso gaussiano, 181, 215, 272

Solucionador de problemas generales, 10

Generalización, 162

Programación genética, 75

Vamos, 102, 103, 108

Meta, 28

 pila, 28

 estado, 87

Gödel

 Kurt,

 teorema de completitud de 5, 5, teorema de incompletitud de 41, 5, 60

gps, 10

Descenso de gradiente, 244

Búsqueda codiciosa, 96, 97, 201

Término básico, 41

H

Problema de detención, 5

regla de Hebb, 225, 234, 247

 binario, 236

Heurística, 46, 57, 94

Función de evaluación heurística, 95, 98

Aprendizaje jerárquico, 274.

Red Hopfield, 226, 227, 237

Cláusula Horn, 26, 73

Hugo, 150

I

ID3, 184

IDA -algoritmo, 100, 250

recompensa inmediata, 260

Implicación, 16

Descenso de gradiente incremental, 245

Aprendizaje incremental, 243

Independiente, 119

 condicionalmente, 146, 147, 155

Indiferencia, 126

Variables indiferentes, 126

máquina de inferencia, 42

mecanismo de inferencia, 12

 contenido informativo, 189

Ganancia de información, 186, 189, 215

Resolución de entrada, 46

Interpretación, 16, 33

Profundización iterativa, 92, 100

j

JavaBayes, 150

k

K método vecino más cercano, 178, 180

K-medias, 208

Núcleo, 181, 253

Método kernel, 253 k

método vecino más cercano, 180

CUCHILLO, 185, 211, 212

conocimiento, 12

 base, 12, 18, 145

 consistente, 23

ingeniero, 9, 12

ingeniería, 12

fuentes, 12

L

Suposición de Laplace, 117.

probabilidades de Laplace, 117

Ley de economía, 197

Aprendizaje perezoso, 182, 215

Aprendizaje, 161

 lote, 243 por

 demostración, 275 distribuido,

 274 jerárquico, 274

 incremental, 201, 243

 máquina, 136 multiagente,

 274 refuerzo, 89,

 107, 214

 semisupervisado, 214 supervisado,

 161, 206, 238

TD-, 273

Agente de aprendizaje, 164

Fase de aprendizaje, 164

Tasa de aprendizaje, 225, 244

Mínimos cuadrados, 143, 241, 242, 245

LEXMED, 115, 122, 131, 193

Recursos limitados, 95

Aproximación lineal, 245

Linealmente separable, 169, 170

LABIOS, 68

LISP, 6, 10

Literales, 21

 complementarios, 23

Regresión lineal ponderada localmente, 183

Lógica

 difusa, 29, 64, 115 de

 orden superior, 59, 60

 probabilística, 13, 29

Teórico Lógico, 6, 10

lógicamente válido, 17

Índice

Aprendizaje automático, 134, 161
distancia manhattan, 100, 207
Distribución marginal, 121
Marginación, 120, 121, 123, 155
Proceso de decisión de Markov, 9, 273 no
 determinista, 270 parcialmente
 observable, 261
Procesos de decisión de Markov, 261.
Implicación material, 16, 115, 128
MaxEnt, 115, 126, 129, 131, 134, 149, 156 distribución,
 126
MPD, 261, 263, 273
 determinista, 269 no
 determinista, 270
Memorización, 161
Aprendizaje basado en la memoria, 182, 183
MGU, 44
Árbol de expansión mínimo, 210
modelo, 17
Modo ponente, 22, 30, 114, 125
impulso, 251
Monotónico, 61
Aprendizaje multiagente, 274
Sistemas multiagente, 10
MYCIN, 10, 114, 132

Naive Bayes, 143, 144, 157, 166, 175, 202, 204, 219,
 298 clasificador,
 202, 204
Ingenuo al revés, 74
Ecuación de Navier-Stokes, 274
Clasificación del vecino
 más cercano, 176
 método, 175
Algoritmo de vecino más cercano, 210
negación, 16
La negación como fracaso, 73;
Red neuronal, 6, 10, 180, 181, 221
Redes neuronales
 recurrentes, 226, 231
Neuroinformática, 231
Neurociencia, 3
neurotransmisor, 223
Ruido, 177
Lógica no monótona, 63, 130
Ecuaciones normales, 242
Forma normal
 conjuntiva, 21
 prenex, 37

O

Observable, 89, 103

Navaja de Occam, 197
Algoritmo fuera de línea, 89
Algoritmo en línea, 89
Ontología, 53
O ramas, 70
Ortonormal, 234
Otelo, 102, 103
Reequipamiento, 177, 197, 198, 201, 240, 243, 252, 273

BÚHO, 53

Paradoja, 60
Paramodulación, 47
Proceso de decisión de Markov parcialmente observable, 261

Problema del pingüino, 77
Perceptrón, 10, 169, 170, 224
transición de fase, 230
FOSO, 129, 130, 149, 157
PL1, 13, 32
Planificación, 76
Política, 260
 método de gradiente, 273
 óptimo, 260
POMDP, 261, 273
Poscondición, 52
condición previa, 52
Lógica de predicados,
 5 de primer orden, 13, 32
Preferencia de aprendizaje, 166.
premisa, 26
Lógica
 probabilística,
 13, 63 razonamiento, 7
Probabilidad, 115, 117
 distribución, 118
 reglas, 137
Regla del producto, 120
Verificación del programa, 51
PRÓLOGO, 7, 10, 20, 28, 67
sistema de prueba, 19
Variables de proposición, 15
Cálculo
 proposicional,
 13 lógica, 15
Poda, 192, 198
Pseudoinversa, 236
Regla literal pura, 46, 55.

q

Q-learning, 267, 275
 convergencia, 269

Quickprop, 252	
R	
variable aleatoria, 116	
Prototipado rápido, 79	
CDR, 53	
decisión en tiempo real, 95	
Requisito en tiempo real, 103	
Característica de funcionamiento del receptor, 142	
Reforzamiento	
aprendizaje, 257, 260	
negativo, 260	
positivo, 260	
Resolución, 6, 22	
cálculo, 10, 20 regla,	
22, 30 general,	
22, 43	
SLD, 27	
solvente, 22	
Recompensa descontada,	
260 inmediata, 260, 269	
Gestión de riesgos, 141	
RoboCup, 11, 273	
Robot, 9, 258	
caminando-, 258	
Robótica, 257	
curva ROC, 143, 214	
RPop, 252	
S	
muestra, 185	
Satisfecho, 17	
Diagrama de dispersión, 163	
Puntuación, 132, 143, 203, 219, 242	
Algoritmo	
de búsqueda, 86	
completo, 87	
óptimo, 89	
heurístico, 84	
espacio, 23, 27, 40, 46	
árbol, 86	
desinformado, 84	
Mapas autoorganizados, 255	
árboles semánticos, 28	
web semántica, 53	
Semántica	
declarativa (PROLOG), 70	
procedimental (PROLOG), 70, 73	
Semidecidible, PL1, 59	
Aprendizaje semisupervisado, 214.	
Sensibilidad, 141, 148	
sensores, 9	
Conjunto de estrategia de apoyo, 46, 55	
Función sigmoidea, 225, 241, 246	
Firma, 15	
semejanza, 175	
Recocido simulado, 232	
Cálculo de situación, 63	
Eskolemización, 39	
resolución SLD, 30	
Reutilización de software, 52	
solución, 87	
sonido, 20	
Correo no	
deseado, filtro 204, 11, 204	
Especificidad, 141	
Estado inicial, 87	
Estado, 87, 258, 259	
espacio, 87	
función de transición, 260	
Inducción estadística, 136	
Subobjetivo, 28, 69	
Axioma de sustitución, 36	
Subsunción, 46	
Vector de soporte, 252	
máquina, 252, 272	
Máquina de vectores de soporte, 181	
SVM, ver máquina de vectores de soporte	
T	
función objetivo, 164	
Tautología, 17	
TD	
-error, 271	
-gammon, 273	
-aprendizaje, 271, 273	
Caja didáctica, 275	
Error de diferencia temporal,	
271	
aprendizaje, 271	
Término, 32	
reescritura, 47	
Datos de prueba, 164, 197	
Texto	
clasificación, 204	
minería, 166	
Probador de teoremas, 6, 42, 43, 47, 51, 52	
Datos de entrenamiento, 58, 164, 197	
función de transición, 270	
Cierto, 34	
tabla de verdad, 16	
método, 19	
turing	
Alan, 5	
prueba, 4	
Ejemplo de Polín, 60, 63, 130	

tu	Iteración de valor, 263
unificable, 44	variables, 32
unificación, 44	VDM-SL, 52
Unificador,	Especificación del método de desarrollo de Viena
44 más general, 44	Idioma, 52
Búsqueda de costos uniformes, 90	Diagrama de Voronoi, 177
cláusula unitaria,	W
46 resolución, 46	Robot caminante, 258
Insatisfecho, 17	WAM, 68, 75
 	Warren máquina abstracta, 68
V	watson, 13
Válido, 17, 34	WEKA, 185, 211

Por lo tanto, está claro que es mejor cambiar a la otra puerta.

Ejercicio 7.5 La función de Lagrange dice $L = \sum_{i=1}^n p_i \ln p_i + \lambda (\sum_{i=1}^n p_i - 1)$. Igualando a cero las derivadas parciales con respecto a p_i y λ se obtiene

$$\frac{\partial L}{\partial p_i} = -\ln p_i - 1 + \lambda = 0 \text{ y } \frac{\partial L}{\partial \lambda} = -\sum_{i=1}^n p_i + 1 = 0.$$

La resta de estas dos ecuaciones da como resultado $p_i = p_j$ para todo $i, j \in \{1, \dots, n\}$. Así $p_1 = p_2 = \dots = p_n = 1/n$.

Ejercicio 7.6

Datos de entrada PIT: var $A\{t, f\}$, $B\{t, f\}$;
 $P([A=t]) = 0,5$;
 $P([B=t] \mid [A=t]) = 0,94$; $QP([B=t])$;

Salida de pozo:
 Informe del estado de las consultas
 Nr Consulta de probabilidad de valor de verdad
 1 SIN ESPECIFICAR 7.200e-01 $QP([B=t])$;

Debido a que PIT opera numéricamente, no puede calcular simbólicamente con el parámetros α y β , más bien sólo con números concretos.

Ejercicio 7.7

Si: $p_1 = P(A, B)$, $p_2 = P(A, \neg B)$, $p_3 = P(\neg A, B)$, $p_4 = P(\neg A, \neg B)$

$$\text{Las restricciones son: } p_1 + p_2 = \alpha \quad (11.1)$$

$$p_4 = 1 - \beta \quad (11.2)$$

$$p_1 + p_2 + p_3 + p_4 = 1 \quad (11.3)$$

Se sigue que $p_3 = \beta - \alpha$. De (11.1) inferimos, basados en la indiferencia, que $p_1 = p_2 = \alpha/2$. Así $P(B) = p_1 + p_3 = \alpha/2 + \beta - \alpha = \beta - \alpha/2 = P(A = B) - 1/2 P(A)$. El programa PIT correspondiente dice

var $A\{t, f\}$, $B\{t, f\}$; $P([A=t]) = 0,6$;
 $P([B=t] \mid [A=t]) = 0,94$;
 $QP([B=t])$.

Ejercicio 7.8 La función de Lagrange se lee

$$L = - \sum_{i=1}^4 p_i \ln p_i + \lambda_1 (p_1 + p_2 - \alpha) + \lambda_2 (p_1 + p_3 - \gamma) + \lambda_3 (p_1 + p_2 + p_3 + p_4 - 1). \quad (11.4)$$

Tomando derivadas parciales para p_1 , p_2 , p_3 , p_4 obtenemos

$$\frac{\partial L}{\partial p_1} = -\ln p_1 - 1 + \lambda_1 + \lambda_2 + \lambda_3 = 0, \quad \frac{\partial L}{\partial p_2} = -\ln p_2 - 1 + \lambda_1 = 0, \quad \frac{\partial L}{\partial p_3} = -\ln p_3 - 1 + \lambda_2 = 0, \quad \frac{\partial L}{\partial p_4} = -\ln p_4 - 1 + \lambda_3 = 0. \quad (11.5)$$

$$\frac{\partial L}{\partial p_2} = -\ln p_2 - 1 + \lambda_1 + \lambda_3 = 0, \quad \partial p_2 \quad (11.6)$$

$$\frac{\partial L}{\partial p_3} = -\ln p_3 - 1 + \lambda_2 + \lambda_3 = 0, \quad \partial p_3 \quad (11.7)$$

$$\frac{\partial L}{\partial p_4} = -\ln p_4 - 1 + \lambda_3 = 0 \quad \partial p_4 \quad (11.8)$$

y calcular

$$(11.5)-(11.6): \ln p_2 - \ln p_1 + \lambda_2 = 0, \quad (11.9)$$

$$(11.7)-(11.8): \ln p_4 - \ln p_3 + \lambda_2 = 0, \quad (11.10)$$

de donde se sigue que $p_3/p_4 = p_1/p_2$, que sustituimos inmediatamente en (7.12) en la página 129:

$$\frac{p_2 p_3}{p_4} + p_2 + p_3 + p_4 = 1$$

$$p_2 + p_4 + \frac{p_3}{p_4} + p_3 + p_4 = 1 \quad (11.11)$$

$$(7.10)-(7.11): p_2 = p_3 + \alpha - \gamma$$

que, sustituido por p_2 , da $(p_3 + \alpha - \gamma)(1 + \frac{p_3}{p_4}) + p_3 + p_4 = 1$

$$(7.10) \text{ en } (7.12): \alpha + p_3 + p_4 = 1. \quad (11.12)$$

Así, eliminamos p_4 en (11.8), lo que da como resultado

$$(p_3 + \alpha - \gamma)(1 + 1 - \alpha \frac{p_3}{p_3}) + 1 - \alpha = 1 \quad (11.13)$$

$$(p_3 + \alpha - \gamma)(1 - \alpha - p_3 + p_3) = \alpha(1 - \alpha - p_3) \quad (11.14)$$

$$(p_3 + \alpha - \gamma)(1 - \alpha) = \alpha(1 - \alpha - p_3) \quad (11.15)$$

$$p_3 + \alpha - \gamma - \alpha p_3 - \alpha^2 + \alpha \gamma = \alpha - \alpha^2 - \alpha p_3 \quad (11.16)$$

$$p_3 = \gamma(1 - \alpha). \quad (11.17)$$

Con (11.12) se sigue que $p_4 = (1 - \alpha)(1 - \gamma)$ y con (11.11) obtenemos $p_2 = \alpha(1 - \gamma)$ y $p_1 = \alpha \gamma$.

Ejercicio 7.9

(a)

$$\text{METRO} = \begin{matrix} 0 & 1000 & 10 \\ & 0 & \end{matrix}.$$

(b)

$$m \cdot \frac{\text{pag}}{1 - \text{pag}} = \frac{1000 \cdot (1 - \text{pag})}{10 \cdot \text{pag}}.$$

Debido a que la decisión entre borrar o leer se encuentra estableciendo el mínimo de los dos valores, es suficiente comparar los dos valores: $1000(1 - p) < 10p$, lo que da como resultado $p > 0.99$. En general, para una matriz M = calculamos el umbral $k/(k + 1)$.

Ejercicio 7.10

(a) Como A y B son independientes, $P(A|B) = P(A) = 0.2$. (b) Aplicando la regla de condicionamiento (página 155) obtenemos

$$P(C|A) = P(C|A, B)P(B) + P(C|A, \neg B)P(\neg B) = 0.2 \cdot 0.9 + 0.1 \cdot 0.1 = 0.19.$$

Ejercicio 7.11

(a)

$$\begin{aligned} P(AI) &= P(AI, \text{Bur}, \text{Oreja}) + P(AI, \neg \text{Bur}, \text{Oreja}) \\ &\quad + P(AI, \text{Bur}, \neg \text{Ear}) + P(AI, \neg \text{Bur}, \neg \text{Ear}) \\ &= P(AI|\text{Bur}, \text{Oído})P(\text{Bur}, \text{Oído}) + P(AI|\neg \text{Bur}, \text{Oído})P(\neg \text{Bur}, \text{Oído}) \\ &\quad + P(AI|\text{Bur}, \neg \text{Oído})P(\text{Bur}, \neg \text{Oído}) \\ &\quad + P(AI|\neg \text{Bur}, \neg \text{Oído})P(\neg \text{Bur}, \neg \text{Oído}) \\ &= 0.95 \cdot 0.001 \cdot 0.002 + 0.29 \cdot 0.999 \cdot 0.002 + 0.94 \cdot 0.001 \cdot 0.998 \\ &\quad + 0.001 \cdot 0.999 \cdot 0.998 \\ &= 0.00252, \\ P(J) &= P(J, AI) + P(J, \neg AI) = P(J|AI)P(AI) + P(J|\neg AI)P(\neg AI) \\ &= 0.9 \cdot 0.0025 + 0.05 \cdot (1 - 0.0025) = 0.052, \\ P(M) &= P(M, AI) + P(M, \neg AI) = P(M|AI)P(AI) + P(M|\neg AI)P(\neg AI) = 0.7 \cdot 0.0025 + \\ &\quad 0.01 \cdot (1 - 0.0025) = 0.0117. \end{aligned}$$

(b)

$$\begin{aligned} P(AI|\text{Bur}) &= P(AI|\text{Bur}, \text{Ear})P(\text{Ear}) + P(AI|\text{Bur}, \neg \text{Ear})P(\neg \text{Ear}) \\ &= 0.95 \cdot 0.002 + 0.94 \cdot 0.998 = 0.94002, \\ P(M|\text{Bur}) &= P(M, \text{Bur})/P(\text{Bur}) = [P(M, AI, \text{Bur}) + P(M, \neg AI, \text{Bur})] \\ &\quad /P(\text{fresa}) \\ &= [P(M|AI)P(AI|\text{Bur})P(\text{Bur}) + P(M|\neg AI)P(\neg AI|\text{Bur})P(\text{Bur})] \\ &\quad /P(\text{fresa}) \\ &= P(M|AI)P(AI|\text{Bur}) + P(M|\neg AI)P(\neg AI|\text{Bur}) \\ &= 0.7 \cdot 0.94002 + 0.01 \cdot 0.05998 = 0.659. \end{aligned}$$

(c)

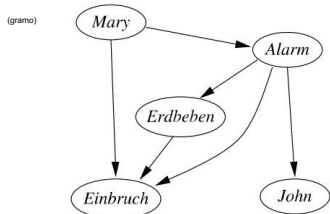
$$P(\text{Rebaba}|M) = \frac{P(M|\text{Rebaba})P(\text{Rebaba})}{P(M)} = \frac{0.659 \cdot 0.001}{0.0117} = 0.056.$$

(d)

$$\begin{aligned} P(AI|J, M) &= \frac{P(AI, J, M)}{P(J, M)} = \frac{P(AI, J, M)}{P(AI, J, M) + P(\neg AI, J, M)} \\ &= \frac{1}{1 + \frac{P(\neg AI, J, M)}{P(AI, J, M)}} = \frac{1}{\frac{P(J|\neg AI)P(M|\neg AI)P(\neg AI)1 + P(J|AI)P(M|AI)P(AI)}{P(AI, J, M)}} \\ &= \frac{1}{1 + \frac{0.05 \cdot 0.01 \cdot 0.9975}{0.9 \cdot 0.7 \cdot 0.0025}} = 0.761, \\ P(J, M|Bur) &= P(J, M|AI)P(AI|Bur) + P(J, M|\neg AI)P(\neg AI|Bur) \\ &= P(J|AI)P(M|AI)P(AI|Bur) \\ &\quad + P(J|\neg AI)P(M|\neg AI)P(\neg AI|Bur) \\ &= 0.9 \cdot 0.7 \cdot 0.94 + 0.05 \cdot 0.01 \cdot 0.06 = 0.5922, \\ P(AI|\neg Bur) &= P(AI|\neg Bur, Oído)P(Oído) + P(AI|\neg Bur, \neg Oído)P(\neg Oído) \\ &= 0.29 \cdot 0.002 + 0.001 \cdot 0.998 = 0.00158, \\ P(J, M|\neg Bur) &= P(J, M|AI)P(AI|\neg Bur) + P(J, M|\neg AI)P(\neg AI|\neg Bur) \\ &= P(J|AI)P(M|AI)P(AI|\neg Bur) \\ &\quad + P(J|\neg AI)P(M|\neg AI)P(\neg AI|\neg Bur) \\ &= 0.9 \cdot 0.7 \cdot 0.00158 + 0.05 \cdot 0.01 \cdot 0.998 = 0.00149, \\ P(J, M) &= P(J, M|Bur)P(Bur) + P(J, M|\neg Bur)P(\neg Bur) \\ &= 0.5922 \cdot 0.001 + 0.00149 \cdot 0.999 = 0.00208, \\ P(Bur|J, M) &= \frac{P(Bur)P(J, M|Bur)}{P(J, M)} = \frac{0.001 \cdot 0.5922}{0.00208} = 0.284. \end{aligned}$$

(e) $P(J)P(M) = 0.052 \cdot 0.0117 = 0.00061$, pero $P(J, M) = 0.00208$ (ver arriba).
Por lo tanto $P(J)P(M) \neq P(J, M)$.

(f) Véase la sección. 7.4.5 en la página 149.

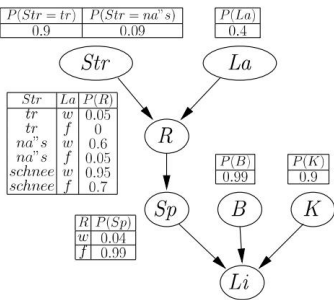


(h) Solo cambia el CPT del nodo de alarma. Todos los demás nodos son independientes del terremoto o independientes de la alarma dada por el terremoto.

(i) Bur P (AI)	
t	0.94
f	0.0016

Otras soluciones son posibles debido al orden variable contraproducente. Así aprendemos: ¡Utilice siempre un orden variable que respete la causalidad!

Fig. 11.1 Red bayesiana para la luz de la bicicleta. El CPT para Li se da en el problema declaración



Ejercicio 7.12

(a), (b) Ver Fig. 11.1.

(c) Falta el borde (Str, Li) porque Li y Str son condicionalmente independientes dado V, es decir, $P(Li | V, Str) = P(Li | V)$ porque la condición de la calle no tiene influencia directa sobre la luz, sino solo sobre la dinamo y el voltaje que genera. (La independencia condicional de Li y Str dada V no puede mostrarse aquí mediante un cálculo basado en los datos disponibles para $P(Li | V, Str)$ y $P(Li | V)$, sino que solo puede afirmarse.) (d) Para el CPT:

$$\begin{aligned} P(R | Str) &= P(R | Str, Flw)P(Flw) + P(R | Str, \neg Flw)P(\neg Flw) \\ &= 0.95 \cdot 0.4 + 0.7 \cdot 0.6 = 0.8, \\ P(V | Str) &= P(V | R)P(R | Str) + P(V | \neg R)P(\neg R | Str) \\ &= 0.04 \cdot 0.8 + 0.99 \cdot 0.2 = 0.23. \end{aligned}$$

11.8 Aprendizaje automático y minería de datos

Ejercicio 8.1

(a) El agente es una función A, que mapea el vector (t1,t2,t3, d1, d2, d3, f1, f2, f3) que consta de tres valores cada uno para temperatura, presión y humedad a una clase valor k {soleado, nublado, lluvioso}. (b) El archivo de datos de entrenamiento consta de una línea del formulario

ti1, ti2, ti3, di1, di2, di3, fi1, fi2, fi3, ki

para cada elemento de datos de entrenamiento. El índice i se ejecuta sobre todos los datos de entrenamiento. Un archivo concreto podría, por ejemplo, comenzar:

17,17,17,980,983,986,63,61,50,soleado
20,22,25,990,1014,1053,65,66,69,soleado
20,22,18,990,979,929,52,60,61,lluvioso

Ejercicio 8.2 Podemos ver la simetría directamente usando la definición de coeficientes de correlación, o de covarianza. Intercambiar i y j no cambia el valor. Para el

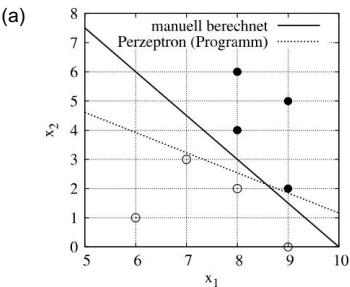
elementos diagonales que calculamos

$$K_{ii} = \frac{\sigma_{ii}}{\sum_{i=1}^n \sigma_{ii}} = \frac{\sum_{p=1}^n (x_{pi} - x_i)(x_{pi} - x_i)}{\sum_{p=1}^n (x_{pi} - x_i)^2} = 1.$$

Ejercicio 8.3 La secuencia de valores para w es

(1, 1), (2,2, -0,4), (1,8, 0,6), (1,4, 1,6), (2,6, 0,2), (2,2, 1,2)

Ejercicio 8.4



(b) Dibujar una línea recta en el gráfico produce:

$$1.5x_1 + x_2 - 15 > 0.$$

Después de 442 iteraciones, los algoritmos de aprendizaje del perceptrón con el vector de inicio $w = (1, 1, 1)$ devoluciones:

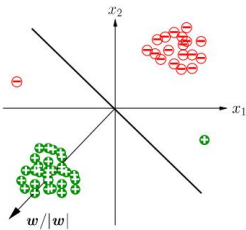
$$w = (11, 16, -129).$$

Esto corresponde a: $0.69x_1 + x_2 - 8.06 > 0$

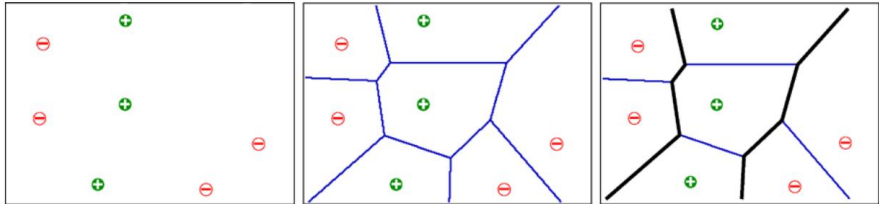
Ejercicio 8.5 (a)

El vector x_{M+} apunta en una dirección "promedio" de todos los puntos positivos y x_{M-} apunta en una dirección "promedio" de todos los puntos negativos. La diferencia de estos vectores apunta desde los puntos negativos hacia los puntos positivos. La línea divisoria es entonces perpendicular a este vector diferencia.

(b) La nube de puntos de los puntos positivos y negativos domina durante el cálculo de w. Los dos mentirosos no juegan un papel aquí. Sin embargo, para determinar la línea divisoria, son importantes.



Ejercicio 8.6



Ejercicio 8.7

- (a) El vecino más cercano es (8, 4), por lo tanto
 clase 1. (b) $k = 2$: clase indefinida ya que una instancia de clase 1 y una instancia de clase 2.
 $k = 3$: decisión 2:1 para clase 0. $k =$
 5: decisión 2:3 para clase 1.

Ejercicio 8.8

- (a) Para poder hacer afirmaciones generales, debemos suponer que los puntos están distribuidos uniformemente en el espacio de características, lo que significa que el número de puntos por área es globalmente aproximadamente igual. Ahora calculamos el área A_d de un anillo angosto de ancho δd y distancia d alrededor del punto x :

$$A_d = \pi (d + \delta d)^2 - \pi d^2 = \pi (d^2 + 2d\delta d + \delta^2 d^2) - \pi d^2 \approx 2\pi d \delta d.$$

- El peso total de todos los puntos en el anillo de ancho δd y distancia d es entonces proporcional a $d \omega_i = d/(1 + \alpha d^2) \approx 1/(\alpha d)$ para $d \rightarrow \infty$. (b) Para esta ponderación, el peso total de todos los puntos en el anillo de ancho δd y $d = d/\delta d = 1$. Así, la distancia d sería proporcional a $d \omega_i$ cada anillo de ancho δd tendría el mismo peso, independientemente de su distancia al punto x . Esto ciertamente no tiene sentido porque los alrededores inmediatos son los más importantes para la aproximación.

Ejercicio 8.9 $\lim_{x \rightarrow 0} x \log_2 x = \lim_{x \rightarrow 0} 0 = \lim_{x \rightarrow 0} 1/x \cdot \frac{\log_2 x}{1/x} = \lim_{x \rightarrow 0} \frac{1/x \cdot (1/x \ln 2)}{1/x^2} = \lim_{x \rightarrow 0} \frac{-1/x^2 \ln 2}{-2/x^3} = \lim_{x \rightarrow 0} \frac{-x}{2 \ln 2} =$
 ecuación se utilizó la regla de l'Hospital.

Ejercicio 8.10 (a) 0 (b) 1 (c) 1.5 (d) 1.875 (e) 2.32 (f) 2

Ejercicio 8.11

- (a) De $\log_2 x = \ln x / \ln 2$, se sigue que $c = 1 / \ln 2 \approx 1.44$. (b) Dado que ambas funciones de entropía solo difieren en un factor constante, la posición del máximo de entropía no cambia. Extrema bajo apremios también mantiene la misma posición. Por tanto, el factor c no causa ningún problema para el método Max Ent. Para el aprendizaje de los árboles de decisión, se comparan las ganancias de información de varios atributos. Debido a que aquí solo importa el orden, pero no el valor absoluto, el factor c tampoco causa ningún problema aquí.

Ejercicio 8.12

- (a) Para el primer atributo calculamos

$$G(D, x_1) = H(D) - \sum_{y_0=6}^9 \frac{|D_{x_1=y_0}|}{|D|} H(D_{x_1=y_0})$$

$$= 1 - \frac{1}{8} H(D_{x_1=6}) + \frac{1}{8} H(D_{x_1=7}) + \frac{3}{8} H(D_{x_1=8}) + \frac{3}{8} H(D_{x_1=9})$$

alto (diámetro $x_1 = 9$)

66,6 % de corrección de los datos de entrenamiento
100 % de corrección de los datos de prueba
80 % de corrección total

Ejercicio 8.14

- (a) Ecuación (8.7) en la página 189 para calcular la ganancia de información de un atributo un lee

$$\text{InfoGanancia}(D, A) = H(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i),$$

donde n es el número de valores del atributo actualmente observado. Si el atributo A se prueba en algún lugar del subárbol como sucesor del valor a_j , entonces solo el valor $A = a_j$ ocurrirá en el conjunto de datos D_j y cada uno de sus subconjuntos D' . Por lo tanto, $D' = D_j$ y para todo $k \neq j$, $|D'_k| = 0$ y obtenemos

$$\text{InfoGanancia}(D', A) = H(D') - \sum_{i=1}^n \frac{|D'_i|}{|D'|} H(D'_i) = H(D'_j) - \frac{|D'_j|}{|D'|} H(D'_j) = 0.$$

El atributo repetido tiene así una ganancia de información de cero, por lo que ya no se utiliza.

(b) De cada atributo

continuo A se genera un atributo binario $A > \Theta, A$.

Si, más abajo en el árbol, el atributo A se discretiza nuevamente con un umbral diferente Θ' , A , entonces el atributo $A > \Theta', A$ es diferente de $A > \Theta, A$. Si luego tiene una ganancia de información más alta que todos los demás atributos, se utilizará en el árbol. Sin embargo, esto también significa que para múltiples ocurrencias de un atributo continuo, los umbrales deben ser diferentes.

Ejercicio 8.15

- (a) $P(\text{Cielo} = \text{despejado}) = 0,65$
 $(\text{Barra} = \text{ascendente}) = 0,67$

Cielo	Bar	P (Prec = seco Cielo, Barra)
Claro	Ascendente	0,85
Claro	Descendente	0,44
Nublado	Descendente	0,45
Nublado	Descendente	0,2

- (b) $P(\text{Cielo} = \text{despejado}) = 0,65$

Bar	P (Prec = seco Barra)
Ascendente	0,73
Descendente	0,33

Cielo	P (Barra = ascendente Cielo)
Claro	0,72
Nublado	0,57

- (c) Los CPT necesarios para $P(\text{Prec}|\text{Sky}, \text{Bar})$ y $P(\text{Bar}|\text{Sky})$, así como para $P(\text{Sky})$ ya son conocidos.

- (d) $P_a = (0,37, 0,065, 0,095, 0,12, 0,11, 0,13, 0,023, 0,092)$

$P_b = (0,34, 0,13, 0,06, 0,12, 0,15, 0,054, 0,05, 0,1)$

$P = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$ (distribución original) Cuadrática

distancia: $d_q(P_a, P) = 0,0029$, $d_q(P_b, P) = 0,014$ Dist. Kullback-

Leibler: $dk(P_a, P) = 0,017$, $dk(P_b, P) = 0,09$ Ambas métricas de distancia

muestran que la red (a) aproxima mejor la distribución que la red (b). Esto significa que la suposición de que Prec y Sky son condicionalmente independientes dada Bar es menos probable que la suposición de que Sky y Bar son independientes.

(e) $PC = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$. Esta distribución es exactamente igual a la distribución original P . Esto no es sorprendente porque no faltan aristas en la red. Esto significa que no se supusieron independencias.

Ejercicio 8.16 Podemos ver inmediatamente que puntuaciones y perceptrones son equivalentes comparando sus definiciones. Ahora para la equivalencia a Naive Bayes: Primero establecemos que $P(K|S_1, \dots, S_n) > 1/2$ es equivalente a $P(K|S_1, \dots, S_n) > P(\neg K|S_1, \dots, S_n)$ porque $P(\neg K|S_1, \dots, S_n) > 1 - P(K|S_1, \dots, S_n)$. De hecho, estamos tratando con un clasificador bayesiano ingenuo binario aquí.

Aplicamos el logaritmo a la ingenua fórmula de Bayes

$$P(K|S_1, \dots, S_n) = P \frac{P(S_1|K) \cdots P(S_n|K) \cdot P(K)}{P(S_1, \dots, S_n)},$$

y obtener

$$\begin{aligned} \log P(K|S_1, \dots, S_n) &= \log P(S_1|K) + \cdots + \log P(S_n|K) + \log P(K) \\ &\quad - \log P(S_1, \dots, S_n). \end{aligned} \quad (11.18)$$

Para obtener una puntuación, debemos interpretar las variables $S_1, \dots, S_n$ como variables numéricas con los valores 1 y 0. Fácilmente podemos ver que

$$\log P(S_i|K) = (\log P(S_i = 1|K) - \log P(S_i = 0|K))S_i + \log P(S_i = 0|K).$$

Resulta que

$$\begin{aligned} \log P(S_i|K) &= (\log P(S_i = 1|K) - \log P(S_i = 0|K))S_i \\ &\quad + \log P(S_i = 0|K). \end{aligned}$$

Ahora definimos $w_i = \log P(S_i = 1|K) - \log P(S_i = 0|K)$ y $c = \log P(S_i = 0|K)$ y simplificar

$$\log P(S_i = 0|K)$$

$$\log P(S_i|K) = w_i S_i + c.$$

Sustituyendo en (11.18) obtenemos

$$\log P(K|S_1, \dots, S_n) = \sum_{i=1}^n w_i S_i + c + \log P(K) - \log P(S_1, \dots, S_n).$$

Para la decisión K debe ser el caso, según la definición del clasificador de Bayes, que $\log P(K | S_1, \dots, S_n) > \log(1/2)$. Por lo tanto, debe ser el caso de que

$$w_i S_i + c + \log P(K) - \log P(S_1, \dots, S_n) > \log(1/2)$$

o eso

$$w_i S_i > \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n),$$

con lo que hemos definido una puntuación con el umbral $\Theta = \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n)$. Debido a que todas las transformaciones se pueden invertir, también podemos transformar cualquier puntuación en un clasificador bayesiano. Con eso, se ha demostrado la equivalencia.

Ejercicio 8.17 Tomando el logaritmo de (8.10) en la página 206 da como resultado

$$\log P(I | s_1, \dots, s_n) = \log c + \log P(I) + \sum_{i=1}^n \log P(w_i | I).$$

De este modo, los números negativos muy pequeños se convierten en números negativos moderados. Dado que la función logaritmo crece monótonamente, para determinar la clase maximizamos de acuerdo con la regla

$$I_{\text{Ngenio-Bayes}} = \underset{I \in \{w, f\}}{\operatorname{argmax}} \log P(I | s_1, \dots, s_n).$$

La desventaja de este método es el tiempo de cálculo algo más largo en la fase de aprendizaje para textos grandes. Durante la clasificación el tiempo no aumenta, ya que los valores $\log P(I | s_1, \dots, s_n)$ se pueden guardar durante el aprendizaje.

Ejercicio 8.19 Sea f estrictamente monótonamente creciente, es decir, $x, y \in \mathbb{R}$ $x < y \implies f(x) < f(y)$. Si ahora $d_1(s, t) < d_1(u, v)$, entonces claramente $d_2(s, t) = f(d_1(s, t)) < f(d_1(u, v)) = d_2(u, v)$. Como la inversa de f también es estrictamente monótona, la inversa es verdadera, es decir, $d_2(s, t) < d_2(u, v) \implies d_1(s, t) < d_1(u, v)$. Así se ha demostrado que $d_2(s, t) < d_2(u, v) \iff d_1(s, t) < d_1(u, v)$.

Ejercicio 8.20

$$\begin{aligned} x_1 x_2 &= 4, \text{ y por tanto } d_s(x_1, x_2) = \sqrt[4]{\frac{23 \cdot 26}{26 \cdot 20} x_2 x_3} = 2, \\ \text{tanto } d_s(x_2, x_3) &= 11,4 \quad \text{y por} \\ 23 \cdot 20 \cdot x_1 x_3 &= 1, \text{ y por lo tanto } d_s(x_1, x_3) = \sqrt[4]{\frac{2}{21 \cdot 4}} \end{aligned}$$

Las oraciones 1 y 2 son las más similares a la métrica de distancia d_s .

Ejercicio 8.21 Ayuda para problemas con KNIME: www.knime.org/forum

11.9 Redes Neuronales

Ejercicio 9.1 Queremos demostrar que $f(\Theta + x) + f(\Theta - x) = 1$.

$$f(\Theta + x) = 1 + \frac{1}{\text{mi} - \frac{x}{\tau}} = \frac{\frac{x}{\tau}}{1 + \text{mi} \frac{x}{\tau}}, \quad f(\Theta - x) = 1 + \frac{1}{\text{mi} + \frac{x}{\tau}},$$

$$f(\Theta + x) + f(\Theta - x) = 1.$$

Ejercicio 9.3 Cada patrón guardado en la red tiene un tamaño de n bits. La red tiene un total de $n(n - 1)/2$ pesos. Si reservamos 16 bits por peso y definimos el almacenamiento binario de tamaño $16n(n-1)/2$ como igualmente grande, entonces claramente puede almacenar $N = 8n(n-1)/n = 4(n-1)$ patrones n bits en tamaño. Para n grande obtenemos $N = 4n$ como límite.

Si tomamos el cociente α del número de bits almacenables y el número de celdas de almacenamiento disponibles, como en (9.11) en la página 237, entonces obtenemos el valor 1 para la memoria de lista 2 y el valor $\alpha = 0.146n / (16n(n-1)/2) \approx 0.018$ para la red de Hopfield. El almacenamiento clásico tiene así (para 16 bits por peso), una capacidad aproximadamente 55 veces superior.

Ejercicio 9.4

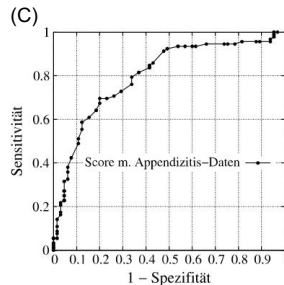
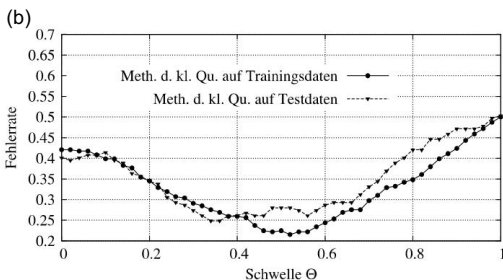
(a) Programa de Mathematica para el método de mínimos cuadrados.

```

MínimoSq[q_,a_] := Módulo[{Nq,Na,m,A,b,w}, Nq = Longitud[q];
m = Longitud[q[[1]]]; Na = Longitud[a]; If[Nq != Na, Print["Longitud[q] != Longitud[a]"];
Salir, 0]; A = Tabla[N[Suma[q[[p,i]] q[[p,j]], {p,1,Nq}]], {i,1,m}, {j,1,m}]; b =
Tabla[N[Suma[a[[p]] q[[p,j]], {p,1,Nq}]], {j,1,m}]; w = Solución Lineal[A,b]

]
LeastSq::usage = "LeastSq[x,y,f] calcula a partir de los vectores de consulta q[[1]],..., q[[m]] una tabla de coeficientes w[[i]]
para un mapeo lineal f[x] = Sum[w[[i]] x[[i]], {i,1,m}] con f[q[[p]]] = a[[p]]".

```



Ejercicio 9.6 (a) El aprendizaje funciona sin errores. (b) ¡El aprendizaje no funciona sin errores!

Ejercicio 9.7

(a) Una función f se llama lineal si para todo x, y, k se da el caso de que $f(x + y) = f(x) + f(y)$ y $f(kx) = kf(x)$. Ahora sean f y g aplicaciones lineales.

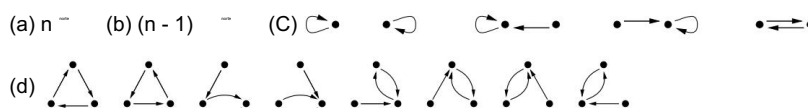
Entonces $f(g(x + y)) = f(g(x) + g(y)) = f(g(x)) + f(g(y))$ y $f(g(kx)) = f(kg(x)) = kf(g(x))$. Por lo tanto, las ejecuciones sucesivas de mapeos lineales son un mapeo lineal.

(b) Observamos dos neuronas de salida arbitrarias j y k . Cada uno de los dos representan una clase. La clasificación se hace formando el máximo de las dos activaciones. i y net_j y net_k en las $\text{net}_j = \sum_i w_{ji} x_i$ ser la suma ponderada de los valores x_i . Sea $\text{net}_j = \text{arrancando}$ neuronas j y k . Además, sea $\text{net}_j > \text{net}_k$. Sin una función de activación, se emite la clase j . Ahora bien, si se aplica una función de activación f estrictamente monótona, nada cambia en el resultado porque, debido a que la función es estrictamente monótona, $f(\text{net}_j) > f(\text{net}_k)$.

Ejercicio 9.8 $f_1(x_1, x_2) = x_1$ espacio l_1^2 , $f_2(x_1, x_2) = x_2$ espacio l_2^2 . Entonces la línea divisoria en el transformado tiene la ecuación $y_1 + y_2 = 1$.

11.10 Aprendizaje por refuerzo

Ejercicio 10.1



Ejercicio 10.2 Iteración de valores con actualizaciones por filas de los valores de V^* de arriba a la izquierda a abajo a la derecha produce

0	0	→	0,81	0,9	→	1,35	1,49	→	2,36	2,62
0	0		0,73	1		1,21	1,66		2,12	2,91

Ejercicio 10.3

(C)

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	1	-1	1	-1	1	-1
5	1	-1	1	-1	1	-1
6	1	-1	1	-1	1	-1

6×6 feedback for a completely smooth surface

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	-1	1	-1	1	-1
3	1	-1	1	-1	1	-1
4	1	-1	1	-1	1	-1
5	1	-1	1	-1	1	-1
6	2	-2	2	-2	2	-2

6×6 feedback for a very heavy surface (like thick snow)

(d) Vemos que cuanto más larga se vuelve una política (es decir, cuantos más pasos, por ejemplo, tiene un ciclo de una estrategia cíclica), más cercano debe estar el valor γ a 1 porque un valor más alto para γ hace que la memoria sea más larga. posible. Sin embargo, la iteración de valor converge mucho más lentamente.

Ejercicio 10.4 El valor V sigue: $(3, 3)$ en la parte inferior derecha de la matriz de estado se cambia como

$V(3, 1) = 0,9 V(2, 1) = 0,9 V(2, 2) = 0,9 V(2, 3) = 0,9 V(3, 3).$ (11.19)

Esta cadena de ecuaciones se sigue de (10.6) porque, para todas las transiciones de estado dadas, la recompensa inmediata máxima es $r(s, a) = 0$, y se da el caso de que

$V(s) = \max_a [r(s, a) + \gamma V(\delta(s, a))] = \gamma V(\delta(s, a)) = 0,9 V(\delta(s, a)).$

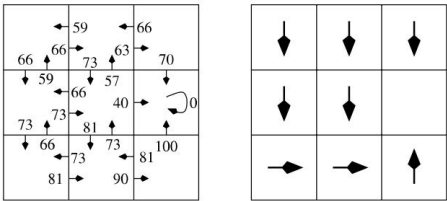
De (10.6) también se sigue que V es $(3, 2) = 1 + 0,9 V(3, 1)$, porque $r(s, a) = 1$ máxima. Análogamente es cierto que $V(3, 3) = 1 + 0,9 V(3, 2)$ y el círculo cierra Las dos últimas ecuaciones juntas producen $V(3, 3) = 1 + 0,9(1 + 0,9 V(3, 3))$. De (11.19) se sigue que $V(3, 3) = 1 + 0,9 V(3, 3)$. Sustituido en $V(3, 3)$, esto

$V(3, 3) = 1 + 0,9(1 + 0,9 V(3, 3)),$

de donde se sigue la demanda.

Ejercicio 10.5

Valores Q estables y una política óptima:



Ejercicio 10.6

(a) Número de estados = ℓ^n . Número de acciones por estado = $\ell^n - 1$. Número de políticas = $(\ell^n)^{\ell^n - 1}$.

(b)

	$\ell = 1$	$\ell = 2$	$\ell = 3$	$\ell = 4$	$\ell = 10$
norte = 1	1	4	27	256	1010
norte = 2	1	256	$3,9 \times 108$	$1,8 \times 1019$	10200
$n = 31$	$1,7 \times 107$	$n = 41$	$1,8 \times 4,4 \times 1038$	$3,9 \times 10115$	103000
1019	$3,9 \times 10154$	$n = 81$	$3,2 \times 10616$	$1,4 \times 1025043$	1040000
6,7	$\times 10315652$				10800000000

(c) Por estado ahora hay $2n$ acciones posibles. Así hay $(2n)^l$ políticas

	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 10$
norte = 1	2	4	8	16	1024
$n = 2$	4	16	64	256	1048576
$n = 4$	16	256	4096	65536	104857600
$n = 8$	64	4096	262144	16777216	10485760000
$n = 16$	256	16384	1048576	67108864	1048576000000

(d) 10120411998 políticas diferentes nunca pueden explorarse combinatoriamente, incluso si todas de las computadoras disponibles en el mundo debían operar en ellas en paralelo. Por lo tanto, los algoritmos "inteligentes" son necesarios para encontrar un óptimo o casi óptimo. política.

Referencias

- [ada75] EW Adams. La lógica de los condicionales. Synthese Library, volumen 86. Reidel, Dordrecht, 1975.
- [Alp04] E. Alpaydin. Introducción al aprendizaje automático. MIT Press, Cambridge, 2004.
- [90 ABRIL] J. Anderson, A. Pellionisz y E. Rosenfeld. Neurocomputación (vol. 2): Direcciones para investigación. MIT Press, Cambridge, 1990.
- [AR88] J. Anderson y E. Rosenfeld. Neurocomputación: Fundamentos de la Investigación. MIT Press, Cambridge, 1988. Colección de artículos originales fundamentales.
- [Barra98] R. Bartak. Guía en línea para la programación de restricciones, 1998. <http://kti.ms.mff.cuni.cz/~bartak/restricciones>.
- [BCDS08] A. Billard, S. Calinon, R. Dillmann y S. Schaal. Programación de robots por demostración. En B. Siciliano y O. Khatib, editores, Handbook of Robotics, páginas 1371–1394. Springer, Berlín, 2008.
- [Bel57] RE Bellman. Programación dinámica. Prensa de la Universidad de Princeton, Princeton, 1957.
- [Ber89] M. Berrondo. Fallgruben für Kopffüssler. Fischer Taschenbuch, Nr. 8703, 1989.
- [BFOS84] L. Breiman, J. Friedman, RA Olshen y CJ Stone. Árboles de clasificación y regresión. Wadsworth, Belmont, 1984.
- [Babero82] W. Bibel. Demostración automática de teoremas. Vieweg, Wiesbaden, 1982.
- [Bis05] CM Obispo. Redes neuronales para reconocimiento de patrones. Prensa de la Universidad de Oxford, Londres, 2005.
- [Bis06] CM Obispo. Reconocimiento de patrones y aprendizaje automático. Springer, Nueva York, 2006.
- [BM03] AG Barto y S. Mahadevan. Avances recientes en el aprendizaje por refuerzo jerárquico. Discrete Event Syst., edición especial sobre aprendizaje por refuerzo, 13:41–77, 2003.
- [Sujetador84] V. Braitenberg. Vehículos—Experimentos en Psicología Sintética. MIT Press, Cambridge, 1984.
- [Sujetador01] B. Brabec. Computergestützte regionale Lawinenprognose. Tesis doctoral, ETH Zürich, 2001.
- [Sujetador11] I. Bratko. PROLOG Programación para Inteligencia Artificial. Addison-Wesley, Lectura, 4ª edición, 2011.
- [Bri91] Enciclopedia Británica. Enciclopedia Británica, Londres, 1991.
- [Bur98] CJ Burgos. Un tutorial sobre máquinas de vectores de soporte para el reconocimiento de patrones. Datos mín. Saber Descubrimiento, 2(2):121–167, 1998.
- [CANALLA] CADE: Jornadas sobre Deducción Automatizada. www.cadeconference.org.
- [Che83] P. Cheeseman. Un método para calcular valores de probabilidad bayesianos generalizados para sistemas expertos. En Proc. del 8º Intl. Conf. conjunta sobre Inteligencia Artificial (IJCAI 83), 1983.
- [Che85] P. Cheeseman. En defensa de la probabilidad. En Proc. del 9º Intl. Conf. conjunta en Inteligencia Artificial (IJCAI-85), 1985.
- [CL73] CL Chang y RC Lee. Demostración de lógica simbólica y teoremas mecánicos. Academic Press, Orlando, 1973.

- [Cle79] WS Cleveland. Robusta de regresión ponderada localmente y diagramas de dispersión de suavizado. *Mermelada. Estadística Asoc.*, 74(368):829–836, 1979.
- [CLR90] T. Cornen, cap. Leiserson y R. Rivest. *Introducción a los Algoritmos*. prensa del MIT, Cambridge, 1990.
- [CM94] WF Clocksin y CS Mellish. *Programación en Prolog*. Springer, Berlín, 4. edición, 1994.
- [Coz98] FG Cozman. *Javabayes, redes bayesianas en java*, 1998. www.cs.cmu.edu/~javabayes.
- [dD91] FT de Dombal. *Diagnóstico del Dolor Abdominal Agudo*. Churchill Livingstone, Londres, 1991.
- [dDLS+72] FT de Dombal, DJ Leaper, JR Staniland, AP McCann y JC Horrocks. Diagnóstico asistido por computadora del dolor abdominal agudo. *Hermano Medicina. J.*, 2:9–13, 1972.
- [Dee11] Proyecto DeepQA, 2011. <http://www.research.ibm.com/deepqa/deepqa.shtml>.
- [DHS01] RO Duda, PE Hart y DG Stork. *Clasificación de patrones*. Wiley, Nueva York, 2001.
- [Diá04] D. Díaz. PRÓLOGO GNU. Universit t Paris, 2004. Aufl. 1.7, para la versi n GNU Prolog 1.2.18, <http://gnu-prolog.inria.fr>.
- [DNM98] CL Blake, DJ Newman, S. Hettich y CJ Merz. Repositorio UCI de la m quina bases de datos de aprendizaje, 1998. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [Ede91] E. Eder. *Complejidades relativas de los c lculos de primer orden*. Vieweg, Wiesbaden, 1991.
- [Alce93] C. Elkan. El  xito parad jico de la l gica difusa. En *Actas de la Und cima Conferencia Nacional sobre Inteligencia Artificial (AAAI-93)*, p ginas 698–703. prensa del MIT, Cambridge, 1993.
- [Ert93] W. Ertel. *Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen*. DISKI, Band 25. Infix, St. Augustin, 1993. Disertaci n, Technische Universit t M nchen.
- [Ert11] W. Ertel. *Inteligencia artificial*, 2011. www.hs-weingarten.de/~ertel/aibook. P gina de inicio de este libro con materiales, programas de demostraci n, enlaces, literatura, fe de erratas, etc.
- [ES99] W. Ertel y M. Schramm. Combinando datos y conocimiento por MaxEnt-optimization de distribuciones de probabilidad. En *PKDD'99 (3ra Conferencia Europea sobre Principios y Pr ctica de Descubrimiento de Conocimiento en Bases de Datos)*. LNCS, tomo 1704, p gs. 323–328. Springer, Praga, 1999.
- [ESCT09] W. Ertel, M. Schneider, R. Cubek y M. Tokic. La caja-ense anza: un universal Marco de aprendizaje de robots. En *Actas de la 14  Conferencia Internacional sobre rob tica avanzada (ICAR 2009)*, 2009. www.servicerobotik.hs-weingarten.de/caja-de-enseanza
- [ESS89] W. Ertel, J. Schumann y Ch. Suttner. Aprendizaje de heur sticas para un probador de teoremas usando propagaci n hacia atr s. En J. Retti y K. Leidlmair, editores, 5. * sterreichische Inteligencia-Artificial-Tagung. Informatik-Fachberichte*, volumen 208, p ginas 87–95. Springer, Berl n, 1989.
- [Ajuste96] Montaje M. *L gica de primer orden y demostraci n autom tica de teoremas*. Springer, Berl n, 1996.
- [FNA+09] D. Ferrucci, E. Nyberg, J. Allan, K. Barker, E. Brown, J. Chu-Carroll, A. Ciccolo, P. Duboue, J. Fan, D. Gondek et al. Hacia el avance abierto de la pregunta sistemas de respuesta. Informe t cnico de IBM RC24789, Yorktown Heights, NY, 2009. http://www.research.ibm.com/deepqa/question_answering.shtml.
- [FPP07] D. Freedman, R. Pisani y R. Purves. *Estad sticas*. Norton, Nueva York, 4  edici n, 2007.
- [Fra05] C. Frayn. Teor a de programaci n de ajedrez por computadora, 2005. www.frayn.net/beowulf/teoria.html.
- [Fre97] E. Freuder. En busca del santo grial. *Restricciones*, 2(1):57–61, 1997.
- [FS97] B. Fischer y J. Schumann. Setheo pasa a la ingenier a de software: aplicaci n de atp a la reutilizaci n de software. En *Jornadas sobre Deducci n Automatizada (CADE-14)*. Incs, volumen 1249, p ginas 65–68. Springer, Berl n, 1997. <http://ase.arc.nasa.gov/people/schumann/publications/papers/cade97-reuse.html>.

- [GW08] RC González y RE Woods. Procesando imagen digital. Pearson/Prentice Hall, Río Upper Saddle/Princeton, 2008.
- [Dios31a] K. Gödel. Discusión sobre Grundlegung der Mathematik, Erkenntnis 2. Monatshefte Mathematik. Phys., 32(1):147–148, 1931.
- [Dios31b] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte Math. Phys., 38(1):173–198, 1931.
- [HKP91] J. Hertz, A. Krogh y R. Palmer. Introducción a la Teoría de la Computación Neural. Addison–Wesley, Reading, 1991.
- [Hon94] B. Hontschik. Theorie und Praxis der Appendektomie. Mabuse, Fráncfort del Meno, 1994.
- [Salto82] JJ Hopfield. Redes neuronales y sistemas físicos con habilidades computacionales colectivas emergentes. proc. nacional Academia ciencia USA, 79:2554–2558, 1982. Reimpresión en [AR88], páginas 460–464.
- [HT85] JJ Hopfield y DW Tank. Cálculo “neuronal” de decisiones en optimización problemas. Biol. Cybern., 52(3):141–152, 1985. Springer.
- [HTF09] T. Hastie, R. Tibshirani y J. Friedman. Los elementos del aprendizaje estadístico: datos Minería, Inferencia y Predicción. Springer, Berlín, 3.ª edición, 2009. Versión en línea: <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>.
- [Jay57] ET Jaynes. Teoría de la información y mecánica estadística. física Rev., 1957.
- [Jay03] ET Jaynes. Teoría de la probabilidad: la lógica de la ciencia. Prensa de la Universidad de Cambridge, Cambridge, 2003.
- [Jen01] FV Jensen. Redes bayesianas y grafos de decisión. Springer, Berlín, 2001.
- [Jor99] MI Jordán, editor. Aprendizaje en Modelos Gráficos. MIT Press, Cambridge, 1999.
- [Kal01] JA Kalman. Razonamiento automatizado con OTTER. Rinton, Paramus, 2001. www-unix.mcs.anl.gov/AR/otter/index.html.
- [Kan89] jue. Kane. Máxima entropía en la lógica probabilística de Nilsson. En Proc. del 11 internacional Conf. conjunta sobre Inteligencia Artificial (IJCAI-89), 1989.
- [KK92] JN Kapur y HK Kesavan. Principios de optimización de entropía con aplicaciones. Prensa Académica, San Diego, 1992.
- [KLM96] LP Kaelbling, ML Littman y AP Moore. Aprendizaje por refuerzo: una encuesta. J. Artif. Intel. Res., 4:237–285, 1996. www-2.cs.cmu.edu/afs/cs/project/jair/pub/volumen4/kaelbling96a.pdf.
- [KMK97] H. Kimura, K. Miyazaki y S. Kobayashi. Aprendizaje por refuerzo en POMDP con aproximación de funciones. En 14th International Conference on Machine Learning, páginas 152–160. Morgan Kaufmann, San Mateo, 1997. <http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robodemo.html>.
- [Koh72] T. Kohonen. Memorias de matrices de correlación. Trans. IEEE. Cómputo., C-21(4):353–359, 1972. Reimpresión en [AR88], páginas 171–174.
- [Lar00] FD Laramée. Programación de ajedrez, partes 1 a 6, 2000. www.gamedev.net/reference/programación/características/ajedrez1.
- [Le999] LEXMED: un sistema experto de aprendizaje para el diagnóstico médico, 1999. www.lexmed.de.
- [Amor78] DW Loveland. Demostración automatizada de teoremas: una base lógica. Holanda Septentrional, Ámsterdam, 1978.
- [LSBB92] R. Letz, J. Schumann, S. Bayerl y W. Bibel. SETHEO: un demostrador de teoremas de alto rendimiento. J. Autom. Reason., 8(2):183–212, 1992. www4.informatik.tu-muenchen.de/~letz/setheo.
- [McC] W. McCune. Sistemas y grupos de deducción automatizados. www-unix.mcs.anl.gov/AR/otros.html. Consulte también www-formal.stanford.edu/cit/ARS/systems.html.
- [McD82] J. McDermott. R1: un configurador de sistemas informáticos basado en reglas. Artefacto Intel., 19:39–88, 1982.
- [MDBM00] G. Melancon, I. Dutour y G. Bousque-Melou. Generación aleatoria de dags para dibujo gráfico. Informe técnico INS-R0005, Centro Holandés de Investigación de Matemáticas y Ciencias de la Computación (CWI), 2000. <http://ftp.cwi.nl/CWIreports/INS/INS-R0005.pdf>.

- [Mit97] T. Mitchell. Aprendizaje automático. McGraw-Hill, Nueva York, 1997. www-2.cs.cmu.edu/~tom/mlbook.html.
- [MP69] M. Minsky y S. Papert. Perceptrones. MIT Press, Cambridge, 1969.
- [Nuevo00] M. Recién nacido. Demostración automatizada de teoremas: teoría y práctica. Springer, Berlín, 2000.
- [Nil86] NJ Nilson. Lógica probabilística. Artefacto Intel., 28(1):71–87, 1986.
- [Nil98] N. Nilsson. Inteligencia artificial: una nueva síntesis. Morgan Kaufmann, San Mateo, 1998.
- [NPW02] T. Nipkow, LC Paulson y M. Wenzel. Isabelle/HOL—Asistente de pruebas para Lógica de orden superior. LNCS, volumen 2283. Springer, Berlín, 2002. www.cl.cam.ac.uk/Research/HVG/Isabelle.
- [NS61] A. Newell y HA Simon. Gps, un programa que simula el pensamiento humano. En H. Billing, editor, Lernende Automaten, páginas 109–124. Oldenbourg, Múnich, 1961.
- [NSS83] A. Newell, JC Shaw y HA Simon. Exploraciones empíricas con la lógica. máquina de la teoría: un estudio de caso en heurística. En J. Siekmann y G. Wrightson, editores, Automatización del razonamiento 1: artículos clásicos sobre lógica computacional 1957–1966, páginas 49–73. Springer, Berlín, 1983. Primera publicación: 1957.
- [OFY+95] C. Ohmann, C. Franke, Q. Yang, M. Margulies, M. Chan, van PJ Elk, FT de Dombal y HD Roher. Diagnosescore für akute Appendizitis. Chirurg, 66:135–141, 1995.
- [OMYL96] C. Ohmann, V. Moustakis, Q. Yang y K. Lang. Evaluación de técnicas automáticas de adquisición de conocimientos en el diagnóstico del dolor abdominal agudo. Artefacto Intel. Med., 8:23–36, 1996.
- [OPB94] C. Ohmann, C. Platen y G. Belenky. Computerunterstützte Diagnostik bei akuten Bauchschmerzen. Chirurg, 63:113–123, 1994.
- [Amigo80] G. Palma. Sobre la memoria asociativa. Biol. Cybern., 36:19–31, 1980.
- [Amigo91] G. Palma. Capacidades de memoria de las reglas locales para la modificación sináptica. Concepts Neu rosci., 2(1):97–128, 1991. MPI Tübingen.
- [Guisante84] J. Perla. Heurísticas, Estrategias de Búsqueda Inteligente para la Resolución de Problemas Informáticos. Addison-Wesley, Reading, 1984.
- [Guisante88] J. Perla. Razonamiento Probabilístico en Sistemas Inteligentes. Redes de Inferencia Plausible. Morgan Kaufmann, San Mateo, 1988.
- [PL05] L. Panait y S. Lucas. Aprendizaje cooperativo multiagente: el estado del arte. Auton. Agentes Multi-Agent Syst., 11(3):387–434, 2005.
- [PS08] J. Peters y S. Schaal. Aprendizaje por refuerzo de la motricidad con gradientes normativos. Neural Netw., 21(4):682–697, 2008. <http://www-clmc.usc.edu/publications/P/peters-NN2008.pdf>.
- [PS09] G. Pólya y S. Sloan. Cómo resolverlo: un nuevo aspecto del método matemático. Ishi, Mountain View, 2009.
- [Qui93] J. Ross Quinlan. C4.5: Programas para Machine Learning. Morgan Kaufmann, San Mateo, 1993. Descarga C4.5: <http://www.rulequest.com/Personal>, Descarga C5.0: <http://www.rulequest.com/download.html>.
- [RB93] M. Riedmiller y H. Braun. Un método adaptativo directo para una retropropagación más rápida aprendizaje: el algoritmo rprop. En Actas de la Conferencia Internacional IEEE on Neural Networks, páginas 586–591, 1993.
- [RGH+06] M. Riedmiller, T. Gabel, R. Hafner, S. Lange y M. Lauer. Troqueles de lluvia de ideas: Entwurfsprinzipien lernfähiger autonomer Roboter. Inform.-Spektrum, 29(3):175–190, 2006.
- [RHR86] DE Rumelhart, GE Hinton y RJ Williams. Aprendizaje de representaciones internas por propagación de errores. En [RM86].
- [Ric83] E. Rico. Inteligencia artificial. McGraw-Hill, Nueva York, 1983.
- [RM86] D. Rumelhart y J. McClelland. Procesamiento distribuido en paralelo, volumen 1. MIT Prensa, Cambridge, 1986.

- [RM96] W. Rödder y C.-H. Meyer. Procesamiento coherente del conocimiento a máxima entropía por ESPÍRITU. En KI-96 (Conferencia Nacional Alemana sobre IA), Dresden, 1996.
- [RMD07] M. Riedmiller, M. Montemerlo y H. Dahlkamp. Aprender a conducir un coche real en 20 minutos. En FBIT '07: Actas de 2007 Frontiers in the Convergence of Biociencia y tecnologías de la información, páginas 645–650. sociedad informática ieeee, Washington, DC, 2007.
- [RMS92] H. Ritter, T. Martínez y K. Schulten. Computación neuronal y autoorganización mapas Addison–Wesley, Reading, 1992.
- [RN10] S. Russell y P. Norvig. Inteligencia artificial: un enfoque moderno. Prentice Hall, Nueva York, 3.^a edición, 2010. 1.^a edición: 1995. <http://aima.cs.berkeley.edu>.
- [Roba] Sitio oficial de Robocup. www.robocup.org.
- [robo] El simulador de fútbol robocup. <http://sserver.sourceforge.net>.
- [robar65] JA Robinson. Una lógica orientada a máquina basada en el principio de resolución. J. ACM, 12(1):23–41, 1965.
- [Roj96] R. Rojas. Redes neuronales: una introducción sistemática. Springer, Berlín, 1996.
- [Ros58] F. Rosenblatt. El perceptrón: un modelo probabilístico para el almacenamiento de información y organización en el cerebro. psicol. Rev., 65:386–408, 1958. Reimpreso en [AR88], páginas 92–114.
- [Ros09] SM Ross. Introducción a la probabilidad y estadística para ingenieros y científicos. Prensa Académica, San Diego, 2009.
- [RW06] CE Rasmussen y CKI Williams. Procesos Gaussianos para Machine Learning. MIT Press, Cambridge, 2006. Versión en línea: <http://www.gaussianprocess.org/gpml/capitulos/>.
- [SA94] S. Schaal y CG Atkeson. Malabares con robots: implementación de sistemas basados en memoria aprendiendo. Sistema de control IEEE. Mag., 14(1):57–71, 1994.
- [Sam59] AL Samuel. Algunos estudios en aprendizaje automático utilizando el juego de damas. IBM J., 1(3):210–229, 1959.
- [SB98] R. Sutton y A. Barto. Aprendizaje reforzado. MIT Press, Cambridge, 1998. www.cs.ualberta.ca/~sutton/book/the-book.html.
- [SB04] J. Siekmann y Ch. Benzmler. Omega: matemática asistida por computadora. En KI 2004: Avances en Inteligencia Artificial. LNAI, volumen 3238, páginas 3–28. Saltador, Berlín, 2004. www.ags.uni-sb.de/~omega.
- [Sch96] M. Schramm. Indifferenz, Unabhängigkeit und maximale Entropie: Eine wahrscheinlichkeitstheoretische Semantik für Nicht-Monotones Schließen. Dissertationen zur Informatik, Band 4. CS, München, 1996.
- [Sch01] J. Schumann. Demostración automatizada de teoremas en ingeniería de software. Saltador, Berlín, 2001.
- [Sch02] S. Schulz. E: un probador de teoremas cerebros. J. AI Commun., 15(2/3):111–126, 2002. www4.informatik.tu-muenchen.de/~schulz/WORK/epruver.html.
- [Sch04] A. Schwartz. Asesino de spam. O'Reilly, Cambridge, 2004. Página de inicio de SpamAssassin: <http://spamassassin.apache.org>.
- [SE90] cap. Suttner y W. Ertel. Adquisición automática de heurísticas de guía de búsqueda. en 10 En t. Conf. sobre Deducción Automatizada. LNAI, volumen 449, páginas 470–484. Saltador, Berlín, 1990.
- [SE00] M. Schramm y W. Ertel. Razonamiento con probabilidades y máxima entropía: el sistema PIT y su aplicación en LEXMED. En K. Inderfurth et al., editor, Operations Research Proceedings (SOR'99), páginas 274–280. Springer, Berlín, 2000.
- [SE10] M. Schneider y W. Ertel. Aprendizaje de robots por demostración con gaussiana local proceso de regresión. En Actas de la Conferencia Internacional IEEE/RSJ sobre Robots y Sistemas Inteligentes (IROS'10), 2010.
- [SET09] T. Segaran, C. Evans y J. Taylor. Programación de la Web Semántica. O'Reilly, Cambridge, 2009.
- [Sho76] EH Shortliffe. Consultas Médicas Informáticas, MYCIN. Holanda Septentrional, Nueva York, 1976.

- [Esp97] Misterios de la Mente. Problema especial. Scientific American Inc., 1997.
- [Esp98] Explorando la Inteligencia. Scientific American Presents, volumen 9. Scientific American Inc., 1998.
- [SR86] T.J. Sejnowski y C.R. Rosenberg. NETtalk: una red paralela que aprende a leer en voz alta. Informe Técnico JHU/EECS-86/01. Informe técnico de ingeniería eléctrica e informática de la Universidad John Hopkins, 1986. Reimpreso en [AR88], páginas 661–672.
- [SS02] S. Schölkopf y A. Smola. Aprendizaje con núcleos: máquinas de vectores de soporte, regularización, optimización y más allá. MIT Press, Cambridge, 2002.
- [SS06] G. Sutcliffe y C. Suttner. El estado de CASC. AI Commun., 19(1):35–48, 2006. Página de inicio de CASC: www.cs.miami.edu/~tptp/CASC.
- [SSK05] P. Stone, R.S. Sutton y G. Kuhlmann. Aprendizaje por refuerzo para robocup soccer keepaway. Comportamiento adaptativo, 2005. www.cs.utexas.edu/~pstone/Papers/bib2html-enlaces/AB05.pdf.
- [St07] J. Estuardo. Cálculo multivariable. Brooks Cole, Florencia, 2007.
- [SW76] C.E. Shannon y W. Weaver. Mathematische Grundlagen der Informationstheorie. Oldenbourg, München, 1976.
- [Tamaño10] C. Szepesvari. Algoritmos para el aprendizaje por refuerzo. Morgan y Claypool, San Rafael, 2010. Borrador disponible en línea: <http://www.ualberta.ca/~szepesva/RLBook.html>.
- [Ted08] R. Tedrake. Control de aprendizaje en números de Reynolds intermedios. En Taller sobre: Robotics Challenges for Machine Learning II, International Conference on Intelligent Robots and Systems (IROS 2008), Niza, Francia, 2008.
- [TEF09] M. Tokic, W. Ertel y J. Fessler. El rastreador, un demostrador en el aula para el aprendizaje por refuerzo. En Actas de la 22ª Conferencia Internacional de Florida Artificial Conferencia de la Sociedad de Investigación de Inteligencia (FLAIRS 09). Prensa AAAI, Menlo Park, 2009.
- [Tes95] G. Tesauro. Aprendizaje de diferencias temporales y td-gammon. común MCA, 38(3), 1995. www.research.ibm.com/massive/tdl.html.
- [Tok06] M. Tokic. Entwicklung eines Lernfähigen Laufroboters. Diplomarbeit Hochschule Ravensburg-Weingarten, 2006. Inklusive Simulationssoftware verfügbar auf www.hs-weingarten.de/~ertel/kibuch.
- [Tur37] A.M. Turing. Sobre números computables, con una aplicación al problema del estéril de Entscheidungsproblem. proc. largo Matemáticas. Soc., 42(2), 1937.
- [Tur50] A.M. Turing. Maquinaria informática e inteligencia. Mente, 59:433–460, 1950.
- [vA06] L. v. Ahn. Juegos con un propósito. Cómputo IEEE. Mag., 96–98, junio de 2006. <http://www.cs.cmu.edu/~biglou/ieee-gwap.pdf>.
- [Wei66] J. Weizenbaum. ELIZA—un programa de computadora para el estudio de la comunicación en lenguaje natural entre el hombre y la máquina. común ACM, 9(1):36–45, 1966.
- [WF01] I. Witten y E. Frank. Procesamiento de datos. Hanser, München, 2001. Biblioteca Java de minería de datos WEKA: www.cs.waikato.ac.nz/~ml/weka.
- [Whi96] J. Whittaker. Modelos Gráficos en Estadística Multivariada Aplicada. Wiley, Nueva York, 1996.
- [Es decir] U. Wiedemann. PhilLex, Lexikon der Philosophie. www.phillex.de/paradoxa.htm.
- [Wie04] J. Wielemaker. SWI-PROLOG 5.4. Universität van Amsterdam, 2004. www.swi-prolog.org.
- [Wik10] Wikipedia, la enciclopedia libre, 2010. <http://en.wikipedia.org>.
- [Ganar] P. Winston. Demostración del juego. <http://www.ai.mit.edu/courses/6.034/f/gamepair.html> Java Applet para Minimax- y Alpha-Beta-Search.
- [Zel94] A. Zell. Simulación Neuronaler Netze. Addison-Wesley, Reading, 1994. Descripción de SNNs y JNNS: www-ra.informatik.uni-tuebingen.de/SNNs.
- [ZSR+99] A. Zielke, H. Sitter, T.A. Rampp, E. Schäfer, C. Hasse, W. Lorenz y M. Rothmund. Überprüfung eines diagnostischen Scoresystems (puntaje de Ohmann) para morir Apendicitis aguda. Chirurg, 70:777–783, 1999.

Índice

A

probabilidad a priori, 122

A -algoritmo, 98, 250

Acción, 86, 87, 260

Función de activación, 224, 234

Actuador, 9

Teoría de la resonancia adaptativa, 255

Admisible, 98

Agente, 3, 9, 257, 259–261, 265, 267, 268, 270, 272, 273
autónomo,

8 basado en

costos, 12

orientado a costos,

140 distribuido,

8 basado en

objetivos, 9

hardware, 9

inteligente, 9 aprendizaje,

8, 12, 164

reflex, 9

software, 9 basado

en utilidades, 12 con memoria, 9

Agentes, distribuidos, 12

IA, 1

Alarma-ejemplo, 145

Poda alfa-beta, 103

y ramas, 70

Y-o árbol, 70

Apendicitis, 121, 131

Aproximación, 164, 180

ARTE, 255

inteligencia artificial, 1

Memoria asociativa, 232

Atributo, 106, 185

Memoria autoasociativa, 226, 233, 238

B

Chaquete, 108, 273

Retropropagación, 246, 265, 275 regla
de aprendizaje, 247

Retrocediendo, 69, 91

Encadenamiento hacia atrás, 27

Aprendizaje por lotes, 201, 243

Fórmula

de Bayes, véase el teorema del

teorema de Bayes, 122, 155

Fórmula de Bayes, véase el teorema de Bayes

red bayesiana, 7, 10, 64, 115, 144, 146, 184 aprendizaje,
157,

199

BDD, ver diagrama de decisión binaria

Ecuación

de Bellman, 263

principio, 263

unidad de polarización, 173

Diagrama de decisión binaria, 29

Máquina Boltzmann, 232

ciencia del cerebro, 3

vehículo Braitenberg, 1, 8, 179

Factor de ramificación, 83, 87

promedio, 84

efectivo, 87

Predicado incorporado, 76

C C4.5, 184, 200

Cálculo, 20

Gentzen, 40

razonamiento natural, 40

secuencial,

40 CART, 184, 198

CASC, 48

Base de casos,

183 Razonamiento basado en

casos, 183 CBR, ver razonamiento

basado en casos Factores

de certeza, 114 Regla de cadena para redes bayesianas,
120,

154, 155

Chatterbot, 5 Checkers, 102, 103, 108

Ajedrez, 102, 103, 105, 108	Lógica por defecto, 63
Iglesia, Alonso, 5	regla por defecto, 63
Clasificación, 162	Regla delta, 243–245
Clasificador, 162, 164, 245	generalizada, 247
Cláusula, 21	Demodulación, 46
-cabeza, 26	Teoría de Dempster-Schäfer, 115
definida, 26	Gráfico de dependencia, 137
fórmula cerrada, 32	Límite de profundidad, 91
CLP, véase programación lógica de restricciones	Derivación, 20
racimo, 206	Determinista, 89, 102
Agrupación, 206, 216	Sistema de diagnosis, 132
jerárquica, 209	Disyunción, 16, 21
CNF, 21	Distancia métrica, 207
ciencia cognitiva, 3	Inteligencia artificial distribuida, 8
complementaria, 23	Aprendizaje distribuido, 274
completo, 20	Distribución, 118, 134
Diagnóstico informático, 144	Programación dinámica, 263
Conclusión, 26	
Probabilidad condicional, tabla 119,	mi
ver CPT	Aprendizaje electrónico, 5
Condicionamente independiente, 146, 147, 155	Ánimo de aprender, 182, 215.
Acondicionamiento, 149, 155	Evento elemental, 115
Matriz de confusión, 11, 212	Isabel, 5
Conjunción, 16, 21	algoritmo EM, 201, 208
Forma normal conjuntiva, 21	Entropía, 188
Conexionismo, 7	máximo, 115, 122
consistente, 23	Entorno, 9, 12 continuo,
constante, 32	12 determinista, 12
Programación lógica de restricciones, 78	discreto, 12 no
Problema de satisfacción de restricciones, 78	determinista,
Correlación, 137	12 observable, 12
coeficiente, 168	parcialmente
matriz, 216	observable, 12
Función de estimación de costos, 96	Ecuación, dirigida, 47
Función de costo, 87, 98	equivalencia, 16
Matriz de costos, 136, 139	Función de evaluación, 103
CPT, 146, 149, 156, 199, 202	Evento, 115
Cesión de créditos, 107, 260	Sistema experto, 131, 144
Validación cruzada, 198	
Diafonía, 236	F
CSP, 78	hecho, 26
Maldición de la dimensionalidad, 274	Factorización, 23, 45
corte, 71	Falso negativo, 141
D	Falso positivo, 140
separación D, 156	Algoritmo del vecino más lejano, 211
DAG, 155, 200	Característica, 106, 162, 171, 185
DIA, 8	Espacio destacado, 163
Minería de datos, 8, 165, 166, 182, 184, 197	Solucionador de restricciones de dominio finito, 79
Morgan, 37	Oración de primer orden, 32
Decisión, 139	Encadenamiento hacia adelante, 27
Árbol de decisiones, 13, 185	Problema de marco, 63
inducción, 165, 185	Variables libres, 32
aprendizaje, 184, 275	

Este enfoque para resolver problemas de optimización fue introducido como programación dinámica por Richard Bellman ya en 1957 [Bel57]. Bellman reconoció que para una política óptima se da el caso de que:

Independientemente del estado inicial s_t y de la primera acción en a_t , todas las decisiones subsiguientes que procedan de cada posible estado sucesor s_{t+1} deben ser óptimas.

Basado en el llamado principio de Bellman, es posible encontrar una política óptima a través de la optimización local de las acciones individuales. Derivaremos este principio para MDP junto con un algoritmo de iteración adecuado.

Se desea una política óptima π que cumple (10.3) en la página 260 y (10.1) en página 260. Reescribimos las dos ecuaciones y obtenemos

$$V(s_t) = \max_{a_t, a_{t+1}, a_{t+2}, \dots} (r(s_t, a_t) + \gamma (r(s_{t+1}, a_{t+1}) + \gamma (r(s_{t+2}, a_{t+2}) + \dots)). \quad (10.4)$$

Dado que la recompensa inmediata $r(s_t, a_t)$ solo depende de s_t y a_t , pero no de los estados y acciones sucesores, la maximización se puede distribuir, lo que en última instancia da como resultado la siguiente caracterización recursiva de V :

$$\begin{aligned} V(s_t) &= \max_{a_t} [r(s_t, a_t) + \gamma \max_{a_{t+1}, a_{t+2}, \dots} (r(s_{t+1}, a_{t+1}) + \gamma (r(s_{t+2}, a_{t+2}) + \dots))] \\ &= \max_{a_t} [r(s_t, a_t) + \gamma V(s_{t+1})]. \end{aligned} \quad (10.5)$$

La ecuación (10.5) resulta de la sustitución $t \rightarrow t+1$ en (10.4). algo escrito más simple:

$$V(s) = \max_a [r(s, a) + \gamma V(s')], \quad (10.6)$$

Esta ecuación implica, al igual que (10.1) en la página 260, que, para calcular $V(s)$, el im recompensa mediata se agrega a la recompensa de todos los estados sucesores, descontados por el factor γ . Si $V(s')$ es conocido, entonces $V(s)$ resulta claramente por operación local simple maximización sobre todas las acciones posibles a en el estado s . Esto corresponde al Bellman principio, por lo que (10.6) también se llama la ecuación de Bellman.

La política óptima $\pi(s)$ lleva a cabo una acción en el estado s que resulta en la valor máximo $V(s)$. De este modo,

$$\pi(s) = \operatorname{argmax}_a [r(s, a) + \gamma V(s')]. \quad (10.7)$$

De la ecuación de recurrencia (10.6) una regla de iteración para aproximar V sigue de una manera sencilla:

$$\hat{V}(s) = \max_a [r(s, a) + \gamma \hat{V}(s')]. \quad (10.8)$$

Para empezar se inicializan los valores aproximados $\hat{V}(s)$ para todos los estados, por ejemplo con el valor cero. Ahora $\hat{V}(s)$ se actualiza repetidamente para cada estado recursivamente recurriendo al valor $\hat{V}(s')$ del mejor estado sucesor. este proceso de calculando $\hat{V}$ se llama iteración de valor y se muestra esquemáticamente en la figura 10.7

Fig. 10.7 El algoritmo para iteración de valor

ITERACIÓN DE VALOR()
Para todo s ∈ S
 V(s) = 0
Repetir
 Para todo s ∈ S
 V(s) = max_a[r(s, a) + γ V(δ(s, a))]
Hasta que V(s) no cambia

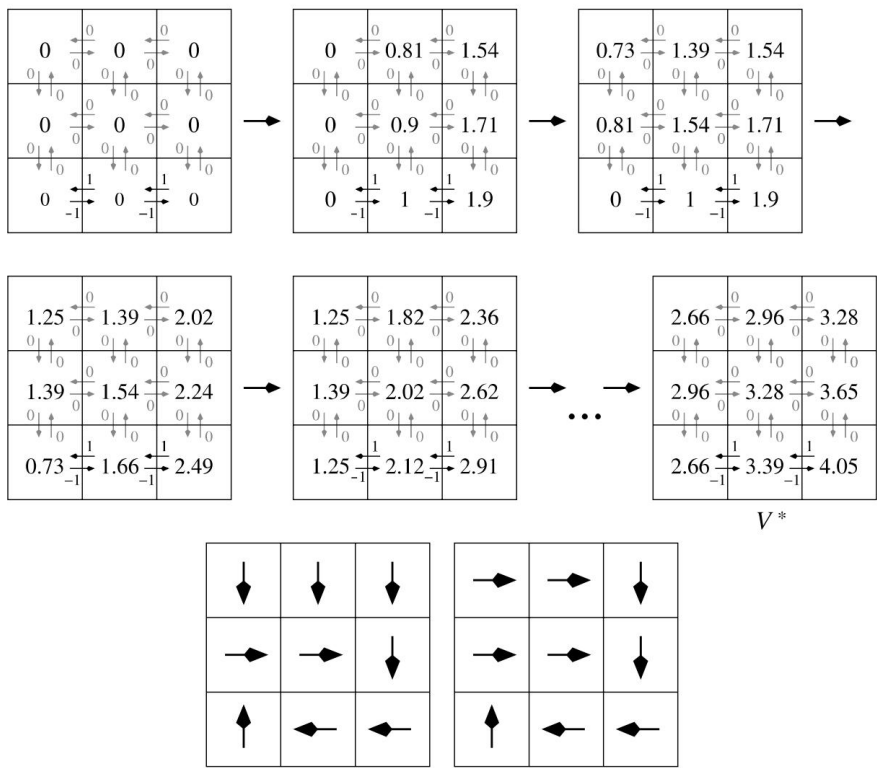


Fig. 10.8 Iteración de valor en el ejemplo con 3 × 3 estados. Las dos últimas imágenes muestran dos óptimos políticas Los números al lado de las flechas dan la recompensa inmediata r(s, a) de cada acción

en la página 264. Se puede demostrar que la iteración de valor converge a V^* [SB98]. Un excelente análisis de algoritmos de programación dinámica se puede encontrar en [Sze10], donde, con base en las propiedades de contracción de los algoritmos particulares (por ejemplo, la iteración de valores), la convergencia se puede probar utilizando la teoría del punto fijo de Banach.

En la figura 10.8, este algoritmo se aplica al ejemplo 10.1 de la página 257 con $\gamma = 0.9$.

En cada iteración, los estados se procesan por filas de abajo a la izquierda a la derecha.

Se muestran varias iteraciones iniciales y en la segunda imagen en la fila inferior los valores límite estables para V

Vemos claramente la progresión del aprendizaje en esta secuencia. El agente explora repetidamente todos los estados, lleva a cabo una iteración de valor para cada estado y guarda la política, que luego se en forma de una función tabulada V tabla de π puede compilar en un archivo. uso eficiente π

Por cierto, para encontrar una política óptima de V sería incorrecto elegir la acción en el estado s_t que resulta en el estado con el máximo V correspondiente a (10.7) en la valor. corre página 263, también se debe agregar la recompensa inmediata $r(s_t, a_t)$ porque estamos buscando V en la Fig. 10.8 en la página 264, esto significa (s_t) y no $V(s_{t+1})$. Aplicado al estado $s = (2, 3)$

$$\begin{aligned}\pi(2, 3) &= \operatorname{argmax}_a [r(s, a) + \gamma V(\delta(s, a))] \\ &= \operatorname{argmax}_{\{izquierda, derecha, arriba\}} \{1 + 0.9 \cdot 2.66, -1 + 0.9 \cdot 4.05, 0 + 0.9 \cdot 3.28\} \\ &= \operatorname{argmax}_{\{izquierda, derecha, arriba\}} \{3.39, 2.65, 2.95\} = izquierda.\end{aligned}$$

En (10.7) en la página 263 vemos que el agente en el estado s_t debe conocer la recompensa inmediata r_t y el estado sucesor $s_{t+1} = \delta(s_t, a_t)$ para elegir la acción óptima en s_t . También debe tener un modelo de las funciones r y δ . Dado que este no es el caso para muchas aplicaciones prácticas, se necesitan algoritmos que también puedan funcionar sin el conocimiento de r y δ . La sección 10.6 está dedicada a dicho algoritmo.

10.5 Un robot que camina aprendiendo y su simulación

En la Fig. 10.9 en la página 266 [TEF09] se muestra una interfaz gráfica de usuario para experimentos simples con aprendizaje por refuerzo. El usuario puede observar el aprendizaje por refuerzo para espacios de estado bidimensionales de diferentes tamaños. Para una mejor generalización, las redes de retropropagación se utilizan para salvar el estado (ver Secc. 10.8). El editor de comentarios que se muestra en la parte inferior derecha, con el que el usuario puede proporcionar manualmente comentarios sobre el entorno, es especialmente interesante para los experimentos. No se muestra el menú para configurar los parámetros para la iteración de valores y el aprendizaje de retropropagación.

Además de la simulación, se desarrollaron específicamente para la enseñanza dos pequeños robots reales que se arrastran con el mismo espacio de estado discreto bidimensional [TEF09]. Los dos robots se muestran en la Fig. 10.10 en la página 266. Cada uno se mueve con un

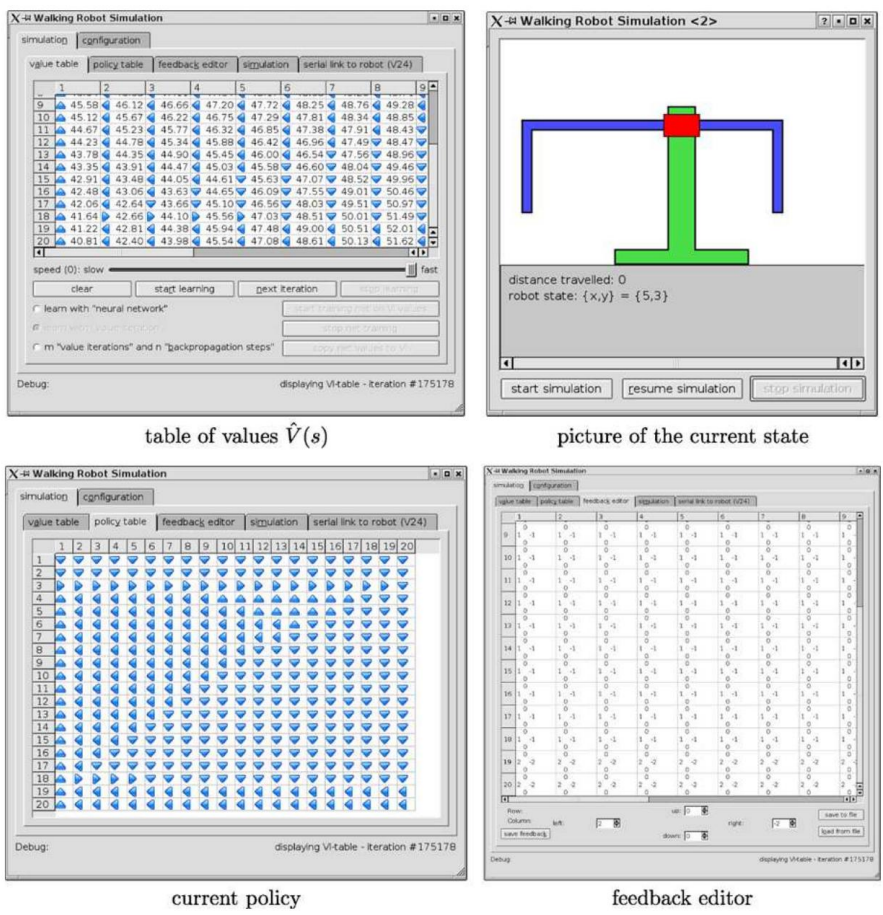


Fig. 10.9 Cuatro ventanas diferentes del simulador de robot andante



Fig. 10.10 Dos versiones del robot que se arrastra

ador Los servos son controlados por un microcontrolador oa través de una interfaz inalámbrica directamente desde una PC. Usando el software de simulación, la matriz de retroalimentación del robot se puede visualizar en la PC. Con estos comentarios guardados, se puede entrenar una política en la PC (que se computa más rápido), luego cargarla nuevamente en el robot y ejecutarla. Sin embargo, el robot también puede aprender de forma autónoma. Para un espacio de estado de tamaño 5×5 , esto toma alrededor de 30 segundos.

Es interesante observar la diferencia entre la simulación y el robot "real". A diferencia de la simulación, el rastreador aprende políticas en las que nunca levanta el brazo del suelo, pero sin embargo avanza de manera muy eficiente. La razón de esto es que, dependiendo de la superficie del subsuelo, la punta de la "axila" puede agarrarse al suelo durante el movimiento hacia atrás, pero se desliza durante el movimiento hacia adelante. Este efecto se percibe muy sensiblemente a través de los sensores de medición de distancia y se evalúa en consecuencia durante el aprendizaje.

La adaptabilidad del robot da como resultado efectos sorprendentes. Por ejemplo, podemos observar cómo el rastreador, a pesar de un servo defectuoso que se desliza en un cierto ángulo, aprende a caminar (más bien cojeando). Incluso es capaz de adaptarse a situaciones cambiantes cambiando las políticas. Un efecto totalmente deseable es la capacidad, dadas superficies lisas diferentes (por ejemplo, diferentes alfombras ásperas), de aprender una política óptima para cada una. También resulta que el robot real es muy adaptable dado un pequeño espacio de estado de tamaño 5×5 .

El lector puede (a falta de un robot real) modelar varias superficies o defectos de servo variando los valores de retroalimentación y luego observando las políticas resultantes (Ejercicio 10.3 en la página 276).

10.6 Aprendizaje Q

Una política basada en la evaluación de posibles estados sucesores claramente no es utilizable si el agente no tiene un modelo del mundo, es decir, cuando no sabe a qué estado conduce una posible acción. En la mayoría de las aplicaciones realistas, el agente no puede recurrir a tal modelo del mundo. Por ejemplo, un robot que se supone que debe sujetar objetos complejos no puede predecir si el objeto se sujetará con seguridad después de una acción de sujeción o si permanecerá en su lugar.

Si no hay un modelo del mundo, se necesita una evaluación de una acción en realizada en el estado s_t , incluso si aún se desconoce a dónde conduce esta acción. Así, ahora trabajamos con una función de evaluación $Q(s_t, a_t)$ para estados con sus acciones asociadas. Con esta función, la elección de la acción óptima se realiza mediante la regla

$$\pi(s) = \underset{a}{\operatorname{argmax}} Q(s, a). \quad (10.9)$$

Para definir la función de evaluación, nuevamente usamos el descuento por pasos de la evaluación para pares estado-acción que ocurren más adelante en el futuro, tal como en (10.1) en la página 260. Por lo tanto, queremos maximizar $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$.

Por lo tanto, para evaluar la acción en el estado s_t definimos en analogía con (10.4) en

página 263:

$$Q(st, at) = \max_{en+1, en+2, \dots} (r(st, at) + \gamma r(st+1, at+1) + \gamma^2 r(st+2, at+2) + \dots). \quad (10.10)$$

De manera análoga al enfoque para la iteración de valores, llevamos esta ecuación a una forma recursiva simple mediante

$$\begin{aligned} Q(st, at) &= \max_{en+1, en+2, \dots} (r(st, at) + \gamma r(st+1, at+1) + \gamma^2 r(st+2, at+2) + \dots) \\ &= r(st, at) + \gamma \max_{en+1, en+2, \dots} (r(st+1, at+1) + \gamma r(st+2, at+2) + \dots) \\ &= r(st, at) + \gamma \max_{en+1} (r(st+1, at+1) + \gamma \max_{en+2} (r(st+2, at+2) + \dots)) \\ &= r(st, at) + \gamma \max_{en+1} Q(st+1, at+1) \\ &= r(st, at) + \gamma \max_{en+1} Q(\delta(st, at), at+1) \\ &= r(s, a) + \gamma \max_{un} Q(\delta(s, a), a'). \end{aligned} \quad (10.11)$$

¿Cuál es entonces la ventaja en comparación con la iteración de valor? La ecuación anterior solo se reescribió ligeramente, pero resulta ser exactamente el enfoque correcto para un nuevo algoritmo. En lugar de guardar V , ahora se guarda la función Q , y el agente puede elegir sus acciones de las funciones δ y r sin un modelo del mundo. Nosotros

Sin embargo, todavía no tenemos un proceso que pueda aprender Q directamente, es decir, sin conocimiento

de V A partir de la formulación recursiva de $Q(s, a)$, se puede derivar un algoritmo de iteración para determinar $Q(s, a)$ en un de manera directa. Inicializamos una tabla $Q(s, a)$ para todos los estados arbitrariamente, por ejemplo con ceros, y realizamos iterativamente

$$\hat{Q}(s, a) = r(s, a) + \gamma \max_{un} \hat{Q}(\delta(s, a), a'). \quad (10.12)$$

Queda por señalar que no conocemos las funciones r y δ . Resolvemos este problema de manera bastante pragmática dejando que el agente en su entorno en el estado s realice la acción a . El estado sucesor es entonces claramente $\delta(s, a)$ y el agente recibe su recompensa del entorno. El algoritmo que se muestra en la Fig. 10.11 en la página 269 implementa este algoritmo para Q-learning.

La aplicación del algoritmo al ejemplo 10.1 en la página 257 con $\gamma = 0.9$ y $n_x = 3$, $n_y = 2$ (es decir, en una cuadrícula de 2×3) se muestra en la figura 10.12 en la página 269 como ejemplo. En la primera imagen, todos los valores de Q se inicializan a cero. En la segunda imagen, después de la primera secuencia de acción, los cuatro valores r que no son iguales a cero se vuelven visibles como valores Q . En la última imagen, se da la política óptima aprendida. El siguiente teorema, cuya demostración se encuentra en [Mit97], muestra que este algoritmo converge no solo en el ejemplo, sino en general.

Q-APRENDIZAJE()

Para todo s S, a A

Q(s, a) = 0 (o al azar)

Repetir

Seleccionar (por ejemplo, aleatoriamente)

un estado

s Repetir Seleccionar una acción a y llevarla a

cabo Obtener recompensa r y nuevo estado s'

Q(s, a) := r(s, a) + γ max_{a'} Q(s', a')

s := s'

Hasta que s es un estado final o límite de tiempo alcanzado

Hasta que Q converge

Fig. 10.11 El algoritmo para Q-learning

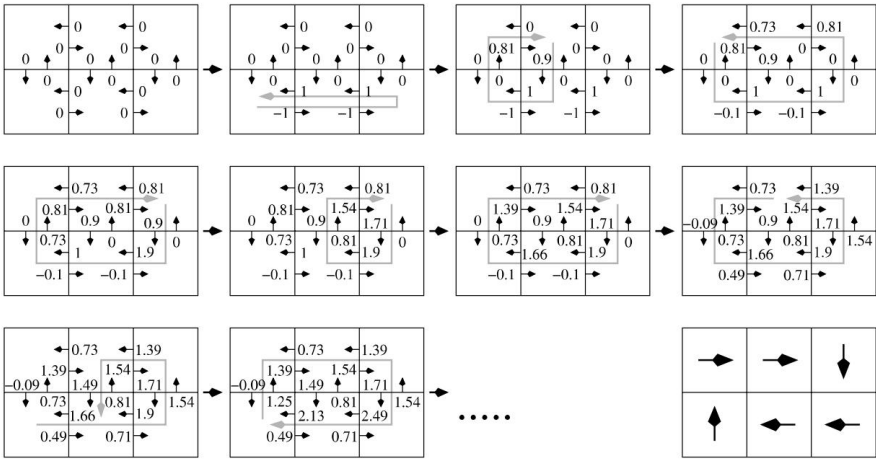


Fig. 10.12 Q-learning aplicado al ejemplo con $n_x = 3$, $n_y = 2$. Las flechas grises marcan las acciones realizadas en cada imagen. Se dan los valores Q actualizados . En la última imagen, se muestra la política actual, que también es óptima.

Teorema 10.1 Sea dado un MDP determinista con recompensa inmediata limitada $r(s, a)$. La ecuación (10.12) en la página 268 con $0 \leq \gamma < 1$ se usa para aprender. Sea $Q^*_n(s, a)$ el valor de $Q(s, a)$ después de n actualizaciones. Si cada par de acciones de estado se visita con una frecuencia infinita, entonces $Q^*_n(s, a)$ converge a $Q(s, a)$ para todos los valores de s y a para $n \rightarrow \infty$.

Prueba Dado que cada transición de estado a acción ocurre infinitamente a menudo, observamos intervalos de tiempo sucesivos con la propiedad de que, en cada intervalo, todas las transiciones de estado a acción ocurren al menos una vez. Ahora mostramos que el error máximo para todas las entradas en la tabla Q^* se reduce por lo menos en el factor γ en cada uno de estos intervalos. Dejar

$$\epsilon_n = \max_{s,a} |Q^n(s, a) - Q(s, a)|$$

sea el error máximo en la tabla Q^n y $s' = \delta(s, a)$. Para cada entrada de la tabla $Q^n(s, a)$ calculamos su contribución al error después de un intervalo como

$$\begin{aligned} |Q^{n+1}(s, a) - Q(s, a)| &= r + \gamma \max_{a'} |Q^n(s', a') - Q(s', a')| \\ &= \gamma \max_{a'} |Q^n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{s', a'} |Q^n(s', a') - Q(s', a')| \leq \\ &\leq \gamma \max_{s, a} |Q^n(s, a) - Q(s, a)| = \gamma \epsilon_n. \end{aligned}$$

La primera desigualdad es verdadera porque, para funciones arbitrarias f y g ,

$$\max_x f(x) - \max_x g(x) \leq \max_x |f(x) - g(x)|$$

y la segunda desigualdad es verdadera porque, por variación adicional del estado s'' , el máximo resultante no puede hacerse más pequeño. Así se ha demostrado que $\epsilon_{n+1} \leq \epsilon_n$. Dado que el error en cada intervalo se reduce por un factor de al menos γ , después de k intervalos es como máximo $\gamma^k \epsilon_0$, y, como resultado, 0 está acotado. Dado que cada estado se visita infinitas veces, hay infinitos intervalos y hasta cero. ϵ_n converge

Según el Teorema 10.1 en la página 269 Q-learning converge independientemente de las acciones elegidas durante el aprendizaje. Esto significa que para la convergencia no importa qué acciones elija el agente, siempre que cada una se ejecute con una frecuencia infinita.

Sin embargo, la velocidad de la convergencia ciertamente depende de los caminos que tome el agente durante el aprendizaje (ver Secc. 10.7).

10.6.1 Q-Learning en un entorno no determinista

En muchas aplicaciones de robótica, el entorno del agente no es determinista. Esto significa que la reacción del medio ambiente a la acción a en el estado s en dos puntos diferentes en el tiempo puede resultar en diferentes estados sucesores y recompensas. Este proceso de Markov no determinista se modela mediante una función de transición probabilística $\delta(s, a)$ y una recompensa inmediata probabilística $r(s, a)$. Para definir la función Q , cada vez se debe calcular el valor esperado sobre todos los posibles estados sucesores. Ecuación (10.11)

en la página 268 se generaliza así a

$$Q(s, a) = E(r(s, a)) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a'), \quad (10.13)$$

donde $P(s' | s, a)$ es la probabilidad de pasar del estado s al estado sucesor s' con la acción a . Desafortunadamente, no hay garantía de convergencia para Q-learning en el caso no determinista si procedemos como antes de acuerdo con (10.12) en la página 268.

Esto se debe a que, en ejecuciones sucesivas a través del ciclo externo del algoritmo de la figura 10.11 en la página 269, el estado de recompensa y sucesor puede ser completamente diferente para el mismo estado s y la misma acción a . Esto puede resultar en una secuencia alterna que salta de un lado a otro entre varios valores. Para evitar este tipo de valores de Q que saltan fuertemente, agregamos el antiguo valor de Q ponderado al lado derecho de (10.12) en la página 268. Esto estabiliza la iteración. La regla de aprendizaje entonces dice

$$Q^n(s, a) = (1 - \alpha_n) Q^{n-1}(s, a) + \alpha_n r(s, a) + \gamma \max_{a'} Q^{n-1}(\delta(s, a), a') \quad (10.14)$$

con un factor de ponderación variable en el tiempo

$$\alpha_n = \frac{1}{1 + \text{mil millones}(s, a)}.$$

El valor $b_n(s, a)$ indica con qué frecuencia se ejecutó la acción a en el estado s en la n -ésima iteración. Para valores pequeños de b_n (es decir, al comienzo del aprendizaje) el término estabilizador $Q^{n-1}(s, a)$ no entra en juego, porque queremos que el proceso de aprendizaje progrese rápidamente. Más tarde, sin embargo, b_n aumenta y, por lo tanto, evita saltos excesivamente grandes en la secuencia de valores de Q . Al integrar (10.14) en Q-learning, los valores $b_n(s, a)$ deben guardarse para todos los pares estado-acción. Esto se puede lograr ampliando la tabla de valores de Q .

Para una mejor comprensión de (10.14), simplificamos esto suponiendo que $\alpha_n = \alpha$ es una constante y transformándolo de la siguiente manera:

$$\begin{aligned} Q^n(s, a) &= (1 - \alpha) Q^{n-1}(s, a) + \alpha r(s, a) + \gamma \max_{a'} Q^{n-1}(\delta(s, a), a') \\ &= Q^{n-1}(s, a) + \alpha r(s, a) + \gamma \underbrace{\max_{a'} Q^{n-1}(\delta(s, a), a') - Q^{n-1}(s, a)}_{\text{TD-error}}. \end{aligned}$$

El nuevo valor de Q $Q^n(s, a)$ se puede representar claramente como el antiguo $Q^{n-1}(s, a)$ más α por un término de corrección que es igual al cambio del valor de Q en este paso. El término de corrección se denomina error TD, o error de diferencia temporal, y la ecuación anterior para cambiar el valor Q es un caso especial de TD-Learning, una clase importante de algoritmos de aprendizaje [SB98]. Para $\alpha = 1$ obtenemos el Q-learning descrito anteriormente. Para $\alpha = 0$, los valores de Q no cambian por completo. Por lo tanto, no se produce ningún aprendizaje.

10.7 Exploración y Explotación

Para Q-learning hasta ahora, solo se ha proporcionado un esquema de algoritmo grueso. Especialmente falta una descripción de la elección del estado inicial cada vez y las acciones a realizar en el bucle interior de la Fig. 10.11 en la página 269. Para la selección de la siguiente acción hay dos posibilidades. Entre las acciones posibles,

uno puede ser elegido al azar. A largo plazo, esto da como resultado una exploración uniforme.

de todas las acciones o políticas posibles, pero con una convergencia muy lenta. Una alternativa

a esto se suma la explotación de los valores de Q^* previamente aprendidos. Aquí el agente siempre

elige la acción con el valor Q^* más alto. Esto da como resultado una convergencia relativamente rápida de una trayectoria específica. Otros caminos, sin embargo, permanecen sin ser visitados en todo el camino.

hasta el final. En el caso extremo entonces podemos obtener políticas no óptimas. En el Teorema 10.1 en la

página 269, por lo tanto, se requiere que cada par estado-acción sea visitado

infinitamente muchas veces. Se recomienda utilizar una combinación de exploración y

explotación con una alta porción de exploración al inicio y reducirla cada vez más.

más con el tiempo.

La elección del estado de partida también influye en la velocidad de aprendizaje. En el primero tres imágenes en la Fig. 10.12 en la página 269 podemos ver claramente que, para las primeras iteraciones, solo se cambian los valores de Q en la vecindad inmediata de los pares estado-acción por recompensa inmediata. Comenzar más lejos de este tipo de punto da como resultado mucho trabajo innecesario. Esto sugiere transferir conocimientos previos sobre la acción estatal se empareja con recompensa inmediata en estados iniciales cercanos a estos puntos. En el curso de aprendizaje, entonces se pueden seleccionar estados iniciales más distantes.

10.8 Aproximación, Generalización y Convergencia

Como se ha descrito Q-learning hasta ahora, se incluye explícitamente una tabla con todos los valores de Q . guardado en una tabla. Esto solo es posible cuando se trabaja con un espacio de estado finito con un número finito de acciones. Sin embargo, si el espacio de estado es infinito, por ejemplo en el caso de variables continuas, entonces no es posible guardar todos los valores de Q ni visitar todos los pares estado-acción durante el aprendizaje.

No obstante, existe una forma sencilla de utilizar Q-learning e iteración de valores en variables continuas. La tabla $Q(s, a)$ se reemplaza por una red neuronal, por ejemplo una red de retropropagación con las variables de entrada s , a y el valor Q como objetivo producción. Por cada actualización de un valor Q , la red neuronal recibe un entrenamiento ejemplo con (s, a) como entrada y $Q(s, a)$ como salida objetivo. Al final tenemos una representación finita de la función $Q(s, a)$. Dado que solo tenemos un número finito de ejemplos de entrenamiento, pero la función $Q(s, a)$ está definida para un número infinito de entradas, por lo tanto, obtendrá automáticamente una generalización si el tamaño de la red se elige adecuadamente (ver Cap. 9). En lugar de una red neuronal, también podemos usar otra supervisada algoritmo de aprendizaje o un aproximador de funciones como una máquina de vectores de soporte o un proceso gaussiano.

Sin embargo, el paso de un número finito de ejemplos de entrenamiento a una función continua puede resultar muy costoso en ciertas situaciones. Q-learning con función de aproximación podría no converger porque el teorema 10.1 en la página 269 solo es cierto si cada par de estado-acción se visita con una frecuencia infinita.

Sin embargo, los problemas de convergencia también pueden surgir en el caso de un número finito de pares estado-acción cuando se usa Q-learning en un POMDP. Q-learning se puede aplicar, en ambas variantes descritas, a procesos de Markov deterministas y no deterministas (MDP). Para un POMDP puede suceder que el agente, debido a sensores ruidosos, por ejemplo, perciba muchos estados diferentes como uno solo. A menudo, muchos estados en el mundo real se asignan a propósito a una supuesta observación. El espacio de observación resultante es entonces mucho más pequeño que el espacio de estado, por lo que el aprendizaje se vuelve más rápido y se puede evitar el sobreajuste (ver Sección 8.4.7).

Sin embargo, al agrupar varios estados, el agente ya no puede diferenciar entre los estados reales, y una acción puede llevarlo a muchos estados sucesores diferentes, según el estado en el que se encuentre realmente. Esto puede conducir a problemas de convergencia de valor. iteración o para Q-learning. En la literatura (por ejemplo, en [SB98]) se sugieren muchos enfoques diferentes para una solución.

También son muy prometedores los llamados métodos de mejora de políticas y sus métodos de gradiente de políticas derivados, en los que los valores de Q no se modifican, sino que la política se modifica directamente. En este esquema se busca una póliza en el espacio de todas las pólizas, que maximiza la recompensa acumulada descontada ((10.1) en la página 260). Una posibilidad de lograr esto es siguiendo el gradiente de la recompensa acumulada al máximo. La política encontrada de esta manera optimiza claramente la recompensa acumulativa. En [PS08] se muestra que este algoritmo puede acelerar mucho el aprendizaje en aplicaciones con grandes espacios de estado, como los que se dan en los robots humanoides.

10.9 Aplicaciones

Mientras tanto, la utilidad práctica del aprendizaje por refuerzo se ha demostrado muchas veces. De un gran número de ejemplos al respecto, presentaremos brevemente una pequeña selección.

TD-learning, junto con una red de retropropagación con 40 a 80 neuronas ocultas, se usó con mucho éxito en TD-gammon, un programa para jugar al backgammon [Tes95]. La única recompensa inmediata del programa es el resultado al final del juego. Una versión optimizada del programa con una anticipación de dos movimientos se entrenó contra sí mismo en 1,5 millones de juegos. Continuó derrotando a jugadores y jugadas de clase mundial, así como a los tres mejores jugadores humanos.

Hay muchas aplicaciones en robótica. Por ejemplo, en RoboCup Soccer Simulation League, los mejores equipos de fútbol de robots ahora utilizan con éxito el aprendizaje por refuerzo [SSK05, Robb]. Equilibrar un poste, que es relativamente fácil para un humano, se ha resuelto con éxito muchas veces con el aprendizaje por refuerzo.

Russ Tedrake hizo una impresionante demostración de la capacidad de aprendizaje de los robots en IROS 2008 en su presentación sobre un modelo de avión que aprende a aterrizar en un punto exacto, como un pájaro que aterriza en una rama [Ted08]. porque el aire

Las corrientes se vuelven muy turbulentas durante una aproximación de aterrizaje tan altamente dinámica, la ecuación diferencial asociada, la ecuación de Navier-Stokes, es irresoluble. Por lo tanto, el aterrizaje no se puede controlar de la manera matemática clásica. El comentario de Tedrake sobre esto:

"¡Los pájaros no resuelven Navier-Stokes!"

Las aves claramente pueden aprender a volar y aterrizar incluso sin la ecuación de Navier-Stokes. Tedrake demostró que esto ahora también es posible para los aviones.

Hoy también es posible aprender a controlar un coche real en tan solo 20 minutos utilizando Q-learning y aproximación de funciones [RMD07]. Este ejemplo muestra que las aplicaciones industriales reales en las que se deben asignar pocas medidas a las acciones se pueden aprender muy bien en poco tiempo.

Los robots reales todavía tienen dificultades para aprender en espacios de acción de estado de alta dimensión porque, en comparación con una simulación, los robots reales obtienen retroalimentación del entorno con relativa lentitud. Debido a las limitaciones de tiempo, los muchos millones de ciclos de entrenamiento necesarios no son, por lo tanto, realizables. Aquí, además de algoritmos de aprendizaje rápido, se necesitan métodos que permitan que al menos partes del aprendizaje sucedan fuera de línea, es decir, sin retroalimentación del entorno.

10.10 Maldición de Dimensionalidad

A pesar del éxito de los últimos años, el aprendizaje por refuerzo sigue siendo un área activa de investigación en IA, sobre todo porque incluso los mejores algoritmos de aprendizaje conocidos en la actualidad siguen siendo poco prácticos para espacios de acción y estados de alta dimensión debido a su gigantesco tiempo de cálculo. Este problema se conoce como la "maldición de la dimensionalidad".

En la búsqueda de soluciones a este problema, los científicos observan animales y humanos durante el aprendizaje. Aquí notamos que el aprendizaje en la naturaleza tiene lugar en muchos niveles de abstracción. Un bebé primero aprende habilidades motoras y de lenguaje simples en el nivel más bajo. Cuando estos se aprenden bien, se guardan y luego se pueden recuperar y usar en cualquier momento. Traducido al lenguaje de la informática, esto significa que cada habilidad aprendida se encapsula en un módulo y luego, en un nivel superior, representa una acción. Al usar acciones tan complejas en un nivel superior, el espacio de acción se reduce considerablemente y, por lo tanto, se acelera el aprendizaje. De manera similar, los estados se pueden abstraer y, por lo tanto, el espacio de estados se puede reducir. Este aprendizaje en múltiples niveles se denomina aprendizaje jerárquico [BM03].

Otro enfoque para la modularización del aprendizaje es el aprendizaje distribuido o aprendizaje multiagente [PL05]. Al aprender las habilidades motoras de un robot humanoide, se deben controlar simultáneamente hasta 50 motores diferentes, lo que da como resultado un espacio de estado de 50 dimensiones y también un espacio de acción de 50 dimensiones. Para reducir esta gigantesca complejidad, el control central se sustituye por un control distribuido. Por ejemplo, cada motor individual podría tener un control individual que lo dirige directamente, si es posible independientemente de los otros motores. En la naturaleza encontramos este tipo de control en los insectos. Por ejemplo, las muchas patas de un milpiés no están dirigidas por un cerebro central, sino que cada par de patas tiene su propio "cerebro" diminuto.

Similar a la búsqueda combinatoria no informada, el aprendizaje por refuerzo tiene la tarea de encontrar lo mejor de una gran cantidad de políticas. La tarea de aprendizaje se vuelve significativamente más fácil si el agente tiene una política más o menos buena antes de que comience el aprendizaje. Entonces las tareas de aprendizaje de alta dimensión se pueden resolver antes. Pero, ¿cómo encontramos esa política inicial? Aquí hay dos posibilidades principales.

La primera posibilidad es la programación clásica. El programador proporciona al agente una política que comprende un programa que considera bueno. Luego se produce un cambio, por ejemplo, a Q-learning. El agente elige, al menos al comienzo del aprendizaje, sus acciones de acuerdo con la política programada y así es conducido a áreas "interesantes" del espacio de acción estatal. Esto puede llevar a reducciones dramáticas en el espacio de búsqueda del aprendizaje por refuerzo.

Si la programación tradicional se vuelve demasiado compleja, podemos comenzar a entrenar al robot o agente haciendo que un humano proscriba las acciones correctas. En el caso más simple, esto se hace mediante el control remoto manual del robot. Luego, el robot guarda la acción proscribida para cada estado y generaliza utilizando un algoritmo de aprendizaje supervisado, como la propagación hacia atrás o el aprendizaje del árbol de decisiones. Este llamado aprendizaje de demostración [BCDS08, SE10] también proporciona una política inicial para el aprendizaje de refuerzo posterior.

10.11 Resumen y perspectiva

Hoy tenemos acceso a algoritmos de aprendizaje establecidos y que funcionan bien para entrenar nuestras máquinas. Sin embargo, la tarea del entrenador o desarrollador humano sigue siendo exigente para aplicaciones complejas. Hay muchas posibilidades sobre cómo estructurar el entrenamiento de un robot y no tendrá éxito sin experimentación. Esta experimentación puede resultar muy tediosa en la práctica porque cada nuevo proyecto de aprendizaje debe ser diseñado y programado. Aquí se necesitan herramientas que, además de los diversos algoritmos de aprendizaje, también ofrezcan al formador la capacidad de combinarlos con la programación tradicional y el aprendizaje de demostración. Una de las primeras herramientas de este tipo es la Teaching-Box [ESCT09], que además de una extensa biblioteca de programas también ofrece plantillas para la configuración de proyectos de aprendizaje y para la comunicación entre el robot y el entorno. Por ejemplo, el maestro humano puede darle al robot más comentarios desde el teclado o a través de una interfaz de voz además de los comentarios del entorno.

El aprendizaje por refuerzo es un área de investigación fascinante y activa que se utilizará cada vez más en el futuro. Cada vez más sistemas de control de robots, pero también otros programas, aprenderán a través de la retroalimentación del entorno. Hoy en día existe una multitud de variaciones de los algoritmos presentados y también algoritmos completamente diferentes. El problema de la escala sigue sin resolverse. Para pequeños espacios de acción y estado con pocos grados de libertad, se pueden lograr resultados impresionantes. Si el número de grados de libertad en el espacio de estados aumenta a 18, por ejemplo para un robot humanoide simple, entonces el aprendizaje se vuelve muy costoso.

Para más lecciones fundamentales, recomendamos la introducción compacta al aprendizaje por refuerzo en el libro de Tom Mitchell [Mit97]. El trabajo estándar de Sutton y Barto [SB98] es minucioso y completo, al igual que el artículo de encuesta de Kaelbling, Littman y Moore [KLM96].

10.12 Ejercicios

Ejercicio 10.1 (a)

Calcule el número de políticas diferentes para n estados y n acciones. Por tanto, son posibles las transiciones de cada estado a cada estado. (b) ¿Cómo cambia el número de políticas en el subproblema (a) si no se permiten acciones vacías, es decir, acciones de un estado a sí mismo? (c) Utilizando diagramas de flechas como los de la figura 10.3 en la página 259, dé todas las políticas para dos estados (d) Usando diagramas de flechas, dé todas las políticas sin acciones vacías para tres estados.

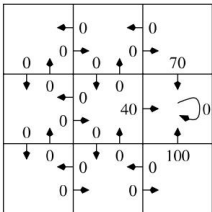
Ejercicio 10.2 Use la iteración de valor manualmente en el Ejemplo 10.1 en la página 257 con $n_x = n_y = 2$.

Ejercicio 10.3 Realice varios experimentos utilizando un simulador de iteración de valores. (a) Instale el simulador de iteración de valor de [Tok06]. (b) Reproduzca los resultados del ejercicio 10.2 introduciendo primero la retroalimentación con el editor de retroalimentación y luego llevar a cabo la iteración de valor. (c) Modele superficies de diferente suavidad y observe cómo cambia la política. (d) Con una matriz de retroalimentación similar, amplíe el espacio de estado gradualmente hasta alrededor de 100×100 y ajuste el factor de descuento γ de modo que resulte una política sensata.

Ejercicio 10.4 Muestre que para el ejemplo de cálculo en la Fig. 10.8 en la página 264 el $V_6(3, 3) \approx 4.05499$, el valor exacto es $V_6(3, 3) = 1,9/(1 - 0,9)$

Ejercicio 10.5

Realice Q-learning en la cuadrícula de 3×3 de la derecha. El estado en el medio a la derecha es un estado meta absorbente.



Ejercicio 10.6 Se da un brazo de robot con n articulaciones (dimensiones) y ℓ estados discretos por articulación. Las acciones de cada estado a cada estado son posibles (si el robot no hace nada, esto se evalúa como una acción (vacía)).

-
- (a) Dé una fórmula para el número de estados y el número de acciones en cada estado y por el número de pólizas para el robot.
- (b) Cree una tabla con el número de estrategias para $n = 1, 2, 3, 4, 8$ y $l = 1, 2, 3, 4, 10$.
- (c) Para reducir el número de estrategias posibles, suponga que el número de acciones posibles por articulación es siempre igual a 2 y que el robot solo puede mover una articulación a la vez. Proporcione una nueva fórmula para el número de estrategias y cree la tabla asociada.
- (d) Con el resultado calculado, justifique que un agente que opera de manera autónoma y adaptativa con $n = 8$ y $l = 10$ ciertamente puede llamarse inteligente.

11.1 Introducción

Ejercicio 1.3 Muchos procesos de inferencia, procesos de aprendizaje, etc. bien conocidos son NP-completos o incluso indecidibles. ¿Qué significa esto para la IA?

Ejercicio 1.4 Si un problema es NP-completo o se puede describir como "difícil", eso significa que hay instancias en las que el problema no se puede resolver en un tiempo aceptable. Este es el llamado peor de los casos. En algunas aplicaciones tenemos que vivir con el hecho de que en el peor de los casos una solución eficiente es imposible. Esto significa que incluso en el futuro habrá problemas prácticamente relevantes que en ciertos casos especiales son irresolubles.

Por lo tanto, la IA no encontrará una fórmula universal ni construirá una supermáquina con la que todos los problemas se puedan resolver. Más bien se da a la tarea de construir sistemas con una mayor probabilidad de encontrar una solución, o con una mayor probabilidad de encontrar soluciones rápidas y óptimas. Nosotros, los humanos en la vida cotidiana, tratamos bastante bien con las soluciones subóptimas. La razón es, sencillamente, el costo excesivo de encontrar la solución óptima. Por ejemplo, solo me toma siete minutos encontrar mi camino del punto A al punto B con un mapa en una ciudad desconocida. El camino más corto habría tomado sólo seis minutos. Sin embargo, encontrar el camino más corto habría llevado tal vez una hora. La prueba de la optimalidad del camino podría ser aún más costosa.

Ejercicio 1.5

(a) La salida depende no sólo de la entrada, sino también del contenido de la memoria.

Para una entrada x , dependiendo del contenido de la memoria, la salida podría ser y_1 o y_2 . Por lo tanto, no es único y, por lo tanto, no es una función. (b)

Si se considera el contenido de la memoria como una entrada adicional, entonces la salida es única (porque el agente es determinista) y el agente representa una función.

Ejercicio 1.6 (a)

Velocidad $v_x(t) = \frac{\partial x}{\partial t}$ límite $\rightarrow 0$ $\frac{x(t) - x(t-t)}{t} \approx \frac{x(t) - x(t-t)}{t}$. se calcula v_y análogamente.

(b) $\frac{\partial^2 x}{\partial t^2} v_x(t) - v_x(t-t)$ (b) Aceleración $\frac{\partial x(t)}{\partial t} = v_x(t) = \lim_{t \rightarrow 0} \frac{x(t) - x(t-t)}{t} = \frac{x(t) - 2x(t-t) + x(t-2t)}{t}$ límite $\rightarrow 0$.
 ay se calcula $\frac{x(t) - 2x(t-t) + x(t-2t)}{t}$ de manera análoga. También se necesita la posición en los tres tiempos $t - 2t$, $t - t$, t .

Ejercicio 1.7

(a) Costos para el agente 1 = $11 \cdot 100$ centavos + $1 \cdot 1$ centavo = 1,101 centavos.Costos para 2 = $0 \cdot 100$ centavos + $38 \cdot 1$ centavo = 38 centavos.Por lo tanto, el agente 2 ahorra 1101 centavos - 38 centavos = $1,063$ centavos.(b) Ganancia para el agente 1 = $189 \cdot 100$ centavos + $799 \cdot 1$ centavo = $19,699$ centavos.Ganancia para el agente 2 = $200 \cdot 100$ centavos + $762 \cdot 1$ centavo = $20,762$ centavos.Por lo tanto, el agente 2 tiene $20,762$ centavos - $19,699$ centavos = $1,063$ centavos más de ganancia.

Si uno evalúa las ganancias perdidas debido a errores, el agente basado en la utilidad toma las mismas decisiones que un agente basado en costos.

11.2 Lógica Proposicional

Ejercicio 2.1 Con la firma $= \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ y la fórmula de variables gramaticales, la sintaxis de la lógica proposicional se puede definir de la siguiente manera:

$$\text{fórmula} ::= \sigma_1 | \sigma_2 | \dots | \sigma_n | \neg f$$

$$| \neg \text{fórmula} | (\text{fórmula}) | \text{fórmula} \rightarrow \text{fórmula}$$

$$| \text{fórmula} \wedge \text{fórmula} | \text{fórmula} \vee \text{fórmula}$$

$$| \text{fórmula} \leftrightarrow \text{fórmula}$$

Ejercicio 2.2 Demostración por el método de la tabla de verdad.

Ejercicio 2.3 (a) $(\neg A \rightarrow B) \rightarrow (\neg B \rightarrow A)$ (b) $(\neg A \rightarrow B) \rightarrow (\neg B \rightarrow A)$ (c) t

Ejercicio 2.4 (a) satisfactorio (b) verdadero (c) insatisfactorio

Ejercicio 2.6

(a) En el Ejercicio 2.3(c) ya se demostró que $A \rightarrow B \rightarrow B$ es una tautología.

El teorema de deducción asegura así la corrección de la regla de inferencia.

(b) Mostramos por el método de la tabla de verdad que $(A \rightarrow B) \rightarrow (\neg B \rightarrow C) \rightarrow (A \rightarrow C)$ es una tautología.

Ejercicio 2.7 La aplicación de la regla de resolución a la cláusula $(f \vee B)$ y $(\neg B \vee f)$ da como resultado el resolvente $(f \vee f) \equiv (f)$. Ahora aplicamos la regla de resolución a las cláusulas B y $\neg B$ y obtenemos la cláusula vacía como resolvente. Porque $(f \vee B) \equiv B$ y $(\neg B \vee f) \equiv \neg B$, $(f) \equiv ()$. Es importante en la práctica que, siempre que se derive la cláusula vacía, se deba a una contradicción.

Ejercicio 2.8 Si KB contiene una contradicción, entonces hay dos cláusulas A y $\neg A$, que permiten derivar la cláusula vacía. La contradicción en KB está claramente todavía en $KB \vdash \neg Q$. Por lo tanto, también permite derivar la cláusula vacía.

Ejercicio 2.9 (a) $(A \vee B) \vee (\neg A \vee \neg B)$ (b) $(A \vee B) \vee (B \vee C) \vee (A \vee C)$

Ejercicio 2.10 Formalización: Cómplice: A, Coche: C, Clave: K

$$WB \equiv (A \vee C) \wedge [(\neg A \vee \neg K) \wedge (A \vee K)] \wedge K$$

Transformación a CNF: $(\neg A \vee \neg K) \wedge (A \vee K) \equiv (\neg K \vee A) \wedge (\neg A \vee K)$

Intenta probar C y agrega $\neg C$ al conjunto de cláusulas. El conjunto de cláusulas CNF es

$$(\neg A \vee C)1 \quad (\neg K \vee A)2 \quad (\neg A \vee K)3 \quad (K)4 \quad (\neg C)5.$$

Prueba de resolución : Res(2, 4) : $(A)6$

Res(1, 6) : $(C)7$

Res(7, 5) : $()8$ Así

se ha demostrado C.

Ejercicio 2.11

(a) $KB \equiv (A \vee B) \wedge (\neg B \vee C)$, $Q \equiv (A \vee C)$

$$KB \quad \neg Q \equiv (A \vee B)1 \quad (\neg B \vee C)2 \quad (\neg A)3 \quad (\neg C)4$$

Prueba de resolución : Res(1, 3) : $(B)5$

Res(2, 4) : $(\neg B)6$

Res(5, 6) : $()$ (b)

$\neg(\neg B \vee (B \vee \neg A) \vee \neg A) \equiv (\neg B)1 \quad (B \vee \neg A)2 \quad (A)3$

Prueba de resolución : Res(1, 2) : $(\neg A)4$

Res(3, 4) : $()$

Ejercicio 2.12 Mediante la aplicación de las equivalencias del Teorema 2.1 de la página 18, podemos demostrar inmediatamente las afirmaciones.

Ejercicio 2.13

Res(8, 9) : $(C \vee F \vee E \vee f)10$

Res(3, 10) : $(F \vee E \vee f)11$

Res(6, 11) : $(A \vee B \vee C \vee E \vee f)12$

Res(1, 12) : $(B \vee C \vee E \vee f)13$

Res(2, 13) : $(C \vee E \vee f)14$

Res(3, 14) : $(E \vee f)15$

Res(5, 15) : $()$

11.3 Lógica de predicados de primer orden

Ejercicio 3.1

(a) x hombre(x) $\neg$ mujer(x) (b) x y z padre(x, y) hombre(x) hijo(y, x, z) (c) x y hermanos(x, y) $[(z$ padre(z, x) padre(z, y)) (z madre(z, x) madre(z, y))] (d) x y z padres(x, y, z) padre(x, z) madre(y, z) (e) x y tío(x, y) z u hijo(y, z, u) hermanos(z, x) varón(x) (f) x y antepasado(x, y) z hijo(y, x, z) u v niño(u, x, v) antepasado(u, y)

Ejercicio 3.2

(a) x y z padre(y, x) madre(z, x) (b) x y hijo(y, x, z) (c) x pájaro(x) moscas (x) (d) x y z animal(x) animal(y) come(x, y) come(y, z) grano(z) (e) x animal(x) (y (come(x, y) (planta(y) (animal(y) z planta(z) come(y, z) mucho_pequeño(y, x)))

Ejercicio 3.3 x y $zx = \text{padre}(y)$ varón(x) hijo(y, x, z)

Ejercicio 3.4

x y $x < y$ $y < x$ $x = y$, x y $x < y$ $\neg y < x$, x y z $x < y$ $y < z$ $x < z$

Ejercicio 3.5

(a) MGU: $x/f(z)$, $u/f(y)$, término: $p(f(z), f(y))$ (b) no

unificable (c) MGU:

$x/\cos y$, $z/4 - 7 \cdot \cos y$, término: $\cos y = 4 - 7 \cdot \cos y$ (d) no unificable (e)

MGU: $u/f(g(w, w))$,

$g(g(w, w), g(w, w))$, $g(g(g(w, w), g(w, w)), g(g(w, w), g(w, w)))$, $x/g(w, w)$, $y/g(g(w, w), g(w, w))$, $z/g(g(g(w, w), g(w, w)), g(g(w, w), g(w, w)))$ término: $q(f(g(w, w), g(g(w, w), g(w, w))), g(g(g(w, w), g(w, w))), g(g(w, w)), g(g(w, w)), g(g(w, w)), g(g(w, w), g(w, w)), g(g(w, w), g(w, w)))$

Ejercicio 3.7

(a) Sea dada la fórmula insatisfactoria $p(x) \neg p(x) r(x)$. Elegimos la cláusula $r(x)$ como el SOS, por lo que no se puede derivar ninguna contradicción. (b) Si el SOS ya es insatisfactorio, entonces no se puede derivar ninguna contradicción. Si no, entonces son necesarios los pasos de resolución entre las cláusulas de SOS y $(KB \neg Q) \text{SOS}$.

- (c) Si no hay complemento a un literal L en una cláusula K, entonces el literal L permanecerá en cada cláusula que se deriva utilizando la resolución de la cláusula K. Por lo tanto, la cláusula vacía no puede derivarse de K o su resolutivo, ni cualquier resolución futura.

Ejercicio 3.8 $\neg Q$ KB $\exists(e = n)$ 1 $(n \cdot x = n)$ 2 $(e \cdot x = x)$ 3 $(\neg a = b)$ 4 Demostración : Dem(1, 2) : $(e \cdot x = e)$ 5 Tra, Sym(3, 5) : $(x = e)$ 6 Dem(4, 6) : $(\neg e = b)$ 7 Dem(7, 6) : $(\neg e = e)$ 8 ()

Aquí "Dem" significa demodulación. La cláusula número 6 se derivó por aplicación de transitividad y simetría de la igualdad en las cláusulas 3 y 5.

Ejercicio 3.9 Los archivos de entrada LOP son:

(a) a;b<- . <- a, b. b;c<- . <- b, c. c;a<- . <-c,a.	(b) <- afeita(púa,X), afeita(X,X). afeita(lengüeta,X); afeita(X,X) <-.	(c) e = norte. <- a = b. m(m(X,Y),Z)=m(X,m(Y,Z)). m(e,X) = X. m(n,X) = n.

11.4 Limitaciones de la lógica

Ejercicio 4.1 (a)

Correctamente: Tomamos un cálculo de prueba completo para PL1. Con él encontramos una prueba para cada fórmula verdadera en tiempo finito. Para todas las fórmulas insatisfactorias, procedo de la siguiente manera: aplico el cálculo a $\neg\phi$ y demuestro que $\neg\phi$ es verdadero. Por lo tanto, ϕ es falso. Así podemos probar cada fórmula verdadera de PL1 y refutar cada fórmula falsa. Desafortunadamente, este proceso no es adecuado para fórmulas satisfactorias.

Ejercicio 4.2 (a)

Se afeita si y sólo si no se afeita. (contradicción) (b) El conjunto de todos los conjuntos que no se contienen a sí mismos. Se contiene a sí mismo si y sólo si no se contiene a sí mismo.

11.5 PRÓLOGO

Ejercicio 5.1 PROLOG señala un desbordamiento de pila. La razón es la primera búsqueda en profundidad de PROLOG, que siempre elige el primer predicado unificable en el archivo de entrada. Con predicados recursivos como la igualdad, esto provoca una recursión sin terminación.

Ejercicio 5.2

```
write_path( [H1,H2|T] ) :- write_path([H2|T]), write_move(H2,H1). escribir_ruta( [_X] ).
```

```
write_move( estado(X,W,Z,K), estado(Y,W,Z,K) ) :-
```

```
    write('Granjero de '), write(X), write(' to '), write(Y), nl.
```

```
write_move( estado(X,X,Z,K), estado(Y,Y,Z,K) ):-
```

```
    write('Granjero y lobo de '), write(X), write(' to '), write(Y),nl.
```

```
write_move( estado(X,W,X,K), estado(Y,W,Y,K) ):-
```

```
    write('Granjero y cabra de '),write(X),write(' to '), write(Y), nl.
```

```
write_move( estado(X,W,Z,X), estado(Y,W,Z,Y) ):-
```

```
    write('Granjero y repollo de '),write(X),write(' to '), write(Y), nl.
```

Ejercicio 5.3

(a) Es necesario dar salida a las soluciones encontradas. La unificación en el plan de hechos (Meta, Meta, Camino, Camino). se encarga de esto. (b)

Las condiciones para la entrada en las cláusulas del predicado seguro se superponen entre sí. El retroceso provocado por la falla conduce a la ejecución de la segunda o tercera alternativa de seguro, donde claramente se volverá a encontrar la misma solución. Un corte al final de las dos primeras cláusulas de seguridad resuelve el problema. Alternativamente, todos los estados seguros se pueden dar explícitamente, como por ejemplo seguro (estado (izquierda, izquierda, izquierda, derecha)).

Ejercicio 5.5 ?- uno(10). uno(0): -

```
    escribir (1). uno(N) :- N1 es N-1,
```

```
    uno(N1), uno(N1).
```

Ejercicio 5.6

(a) Con $n_1 = |L_1|$, $n_2 = |L_2|$ es cierto que $T_{append}(n_1, n_2) = O(n_1)$. (b) Con $n = |L|$

es cierto que $T_{rev}(n) = O(n^2)$. (c) Con $n = |L|$ es

cierto que $T_{acrev}(n) = O(n)$.

Ejercicio 5.7 Para los árboles con símbolos en los nodos internos, podemos usar los términos $a(b,c)$ y $a(b(e,f,g),c(h),d)$, por ejemplo. Árboles sin símbolos: árbol(b,c), o árbol(árbol(e,f,g),árbol(h),d).

Ejercicio 5.8 Programas SWI-PROLOG: (a)

```
fib(0,1). fiba(1,1).
```

```
fib(N,R) :- N1 es N-1, fib(N1,R1), N2 es N-2, fib(N2,R2), R es R1 + R2. (b) T
```

```
(n) = O(2^n) (c) :- dinámica fib/2. fib(0,1).
```

```
fib(1,1).      ^).
```

```
fib(N,R) :- N1 es N-1, fib(N1,R1), N2 es N-2, fib(N2,R2), R es R1 + R2,
```

```
asserta(fib(N,R)).
```

(d) Si los hechos fib(0,1) a fib(k,fib(k)) ya fueron calculados, entonces para la llamada fib(n,X) es cierto que

$$T(n) = \begin{cases} O(1) & \text{si } n \leq k, \\ O(n - k) & \text{si } n > k. \end{cases}$$

(e) Porque después de las primeras n llamadas al predicado fib, todas las llamadas posteriores tienen directo acceso a los hechos.

Ejercicio 5.9

(a) La solución no se revela aquí.

(b)

```
comenzar :-
    fd_domain([británico, sueco, danés, noruego, alemán],1,5),
    fd_all_different([británico, sueco, danés, noruego, alemán]),
    fd_domain([Té, Café, Agua, Cerveza, Leche],1,5),
    fd_all_different([Té, Café, Agua, Cerveza, Leche]),
    fd_domain([Rojo, Blanco, Verde, Amarillo, Azul],1,5),
    fd_all_different([Rojo, Blanco, Verde, Amarillo, Azul]),
    fd_domain([Perro, Pájaro, Gato, Caballo, Pez],1,5),
    fd_all_different([Perro, Pájaro, Gato, Caballo, Pez]),
    fd_domain([Pallmall, Dunhill, Marlboro, Winfield, Rothmann],1,5),
    fd_all_different([Pallmall, Dunhill, Marlboro, Winfield, Rothmann]),
    fd_labeling([británico, sueco, danés, noruego, alemán]),
    Británico #= Rojo, % El británico vive en la casa roja
    Sueco #= Perro, % El Sueco no tiene perro
    Danés #= Té, % Al danés le gusta beber té
    Verde #= Blanco - 1, % La casa verde está a la izquierda de la casa blanca
    Verde #= Café, % El dueño de la casa verde toma café
    Pallmall #= Pájaro, % La persona que fuma Pall Mall tiene un pájaro
    Leche #= 3, % El hombre de la casa del medio bebe leche
    Amarillo #= Dunhill, % El dueño de la casa amarilla fuma Dunhill
    Noruego #= 1, % El noruego vive en la primera casa
    dist(Marlboro,Cat)#= 1, % El fumador de Marlboro vive al lado del gato
    dist(Horse,Dunhill) #= 1, % El hombre con el caballo vive al lado del fumador Dunhill
    Winfield #= Beer, % Al fumador de Winfield le gusta beber cerveza
    dist(Norwegian,Blue) #= 1, % El noruego vive al lado de la casa azul
    German #= Rothmanns, % El alemán fuma Rothmanns
    dist(Marlboro,Water)#=1, % El vecino del fumador de Marlboro bebe agua.
    write([británico, sueco, danés, noruego, alemán]), nl,
    write([Perro, Pájaro, Gato, Caballo, Pez]), nl.
```

11.6 Búsqueda, juegos y resolución de problemas

Ejercicio 6.1

(a) En el último nivel hay b^d nodos. Todos los niveles anteriores juntos tienen

$$Nb(dmáx) = \sum_{yo=0}^{d-1} b^{yo} = \frac{b^d - 1}{b - 1}$$

- nodos, si b se hace grande. porque $b^d / (b - 1) \approx b^d$ hay alrededor de b veces como muchos nodos en el último nivel como en todos los demás niveles juntos.
- (b) Para un factor de ramificación no constante, esta afirmación ya no es cierta, como muestra el siguiente contraejemplo: Un árbol que se ramifica fuertemente hasta el nivel anterior al último, seguido de un nivel con un factor de ramificación constante de 1, tiene exactamente tantos nodos en el último nivel como en el penúltimo nivel.

Ejercicio 6.2

(a) En la Fig. 6.3 en la página 86, la estructura del árbol en el segundo nivel con sus ocho repeticiones de nodos. Por lo tanto, podemos calcular el factor de ramificación promedio b_m a partir de $b_m^2 = 8$ a $b_m = \sqrt{8}$.

- (b) Aquí el cálculo no es tan simple porque el nodo raíz del árbol se ramifica más que todos los demás. Sin embargo, podemos decir que en el interior del árbol el factor de ramificación es exactamente 1 más pequeño de lo que sería sin la verificación del ciclo. Entonces $b_m \approx \sqrt[3]{8 - 1} \approx 1.8$.

Ejercicio 6.3 (a)

Para el factor de ramificación promedio, el número de nodos hoja es fijo. Para el factor de ramificación efectivo, por el contrario, el número de nodos en todo el árbol es fijo.

- (b) Porque el número de todos los nodos en el árbol suele ser una mejor medida del tiempo de cálculo para buscar en un árbol completo que el número de nodos hoja. (c) Para un gran b según (6.1) en la página 87 tenemos $n \approx b^d + 1 / b^d = b^d$, dando

$$b^d = \sqrt[d]{n}.$$

Ejercicio 6.4 (a)

3-rompecabezas: $4! = 24$ estados, 8-rompecabezas: $9! = 362\,880$ estados,

15 rompecabezas: $16! = 20\,922\,789\,888\,000$ estados.

- (b) Después de mover el cuadrado vacío 12 veces en el sentido de las agujas del reloj, alcanzamos nuevamente el estado inicial y creamos así un subespacio cíclico con 12 estados.

Ejercicio 6.6 (a) Programa Mathematica:

```

0 BreadthFirstSearch[Nodo_, Meta_, Profundidad_] := Módulo[{i, NewNodes={}},
1 Para[i=1, i<= Longitud[Nodo], i++, 2
      NewNodes = Join[ NewNodes, Successors[ Nodo[[i]] ] ];
3      If[MiembroQ[Sucesores[ Nodo[[i]] ], Meta],
4          Devolver[{"Solución encontrada", Profundidad+1}], 0
5      ]
6  ];
7 If[Length[NewNodes] > 0, 8
      BreadthFirstSearch[NuevosNodos, Objetivo, Profundidad+1],
9      Retomar[{"Fallo", Profundidad}]
10 ]
11 ]

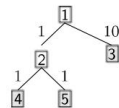
```

- (b) Porque la profundidad del espacio de búsqueda no está limitada.

Ejercicio 6.7 (a)

Para un costo constante 1, el costo de todos los caminos en la profundidad d es menor que el costo de todos los caminos en la profundidad $d + 1$. Dado que todos los nodos en la profundidad d se prueban antes que el primer nodo en la profundidad $d + 1$, se garantiza que se encontrará una solución de longitud d antes que una

solución de longitud $d + 1$. (b) Para el árbol de búsqueda vecino, el nodo número 2 y el nodo de solución número 3 de costo 10 se generan primero. La búsqueda termina. Los nodos 4 y 5 con costos de ruta de 2 cada uno no se generan. Por lo tanto, no se encuentra la solución óptima.



9.3.4 Un programa de corrección ortográfica

Como una aplicación de la memoria asociativa descrita con la regla binaria de Hebb, elegimos un programa que corrige entradas erróneas y las asigna a palabras guardadas de un diccionario. Claramente, aquí se necesitaría una memoria autoasociativa. Sin embargo, debido a que codificamos los vectores de consulta y destino de manera diferente, este no es el caso. Para los vectores de consulta q elegimos una codificación de pares. Para un alfabeto de 26 caracteres hay $26 \cdot 26 = 676$ pares ordenados de letras. Con 676 bits, el vector de consulta tiene un bit para cada uno de los pares posibles

aa, ab,..., az, ba,..., bz,..., za,..., zz.

Si hay un par de letras en la palabra, entonces se ingresará un uno en el lugar apropiado. Para la palabra "henry", por ejemplo, los espacios para "ha", "an" y "ns" se llenan con unos. Para el vector de destino t , se reservan 26 bits para cada posición en la palabra hasta una longitud máxima (por ejemplo, diez caracteres). Para la i -ésima letra del alfabeto en la posición j en la palabra, entonces se establece el número de bit $(j - 1) \cdot 26 + i$. Para la palabra "henry", se establecen los bits 8, 27, 66 y 97. Para un máximo de 10 letras por palabra, el vector objetivo tiene una longitud de 260 bits.

La matriz de peso W tiene así un tamaño de $676 \cdot 260$ bits = 199420 bits, que por

(9.11) en la página 237 puede almacenar como máximo

$$N_{\text{máx}} \leq 0,69 = \overset{\text{Minnesota}}{0,69} \approx 130 \cdot 676 + \overset{676 \cdot 260}{260 \cdot m + n}$$

palabras. Con 72 nombres, ahorramos aproximadamente la mitad y probamos el sistema. Los nombres almacenados y la salida del programa para varias entradas de ejemplo se dan en la Fig. 9.12 en la página 239. El umbral siempre se inicializa al número de bits en la consulta codificada. Aquí este es el número de pares de letras, por lo tanto, la longitud de la palabra menos uno. Luego se reduce gradualmente a dos. Podríamos automatizar aún más la elección del umbral comparándolo con el diccionario para cada intento de umbral y generar la palabra encontrada cuando la comparación tenga éxito.

La reacción a las entradas ambiguas "andr" y "johanne" es interesante. En ambos casos, la red crea una mezcla de dos palabras guardadas que encajan. Vemos aquí una importante fortaleza de las redes neuronales. Son capaces de hacer asociaciones con objetos similares sin una métrica de similitud explícita. Sin embargo, de manera similar a la búsqueda heurística y la toma de decisiones humanas, no hay garantía de una solución "correcta".

Dado que los datos de entrenamiento deben estar disponibles en forma de pares de entrada-salida para todos los modelos neuronales que se han introducido hasta ahora, estamos tratando con aprendizaje supervisado, que también es el caso de las redes que se presentan en las siguientes secciones.

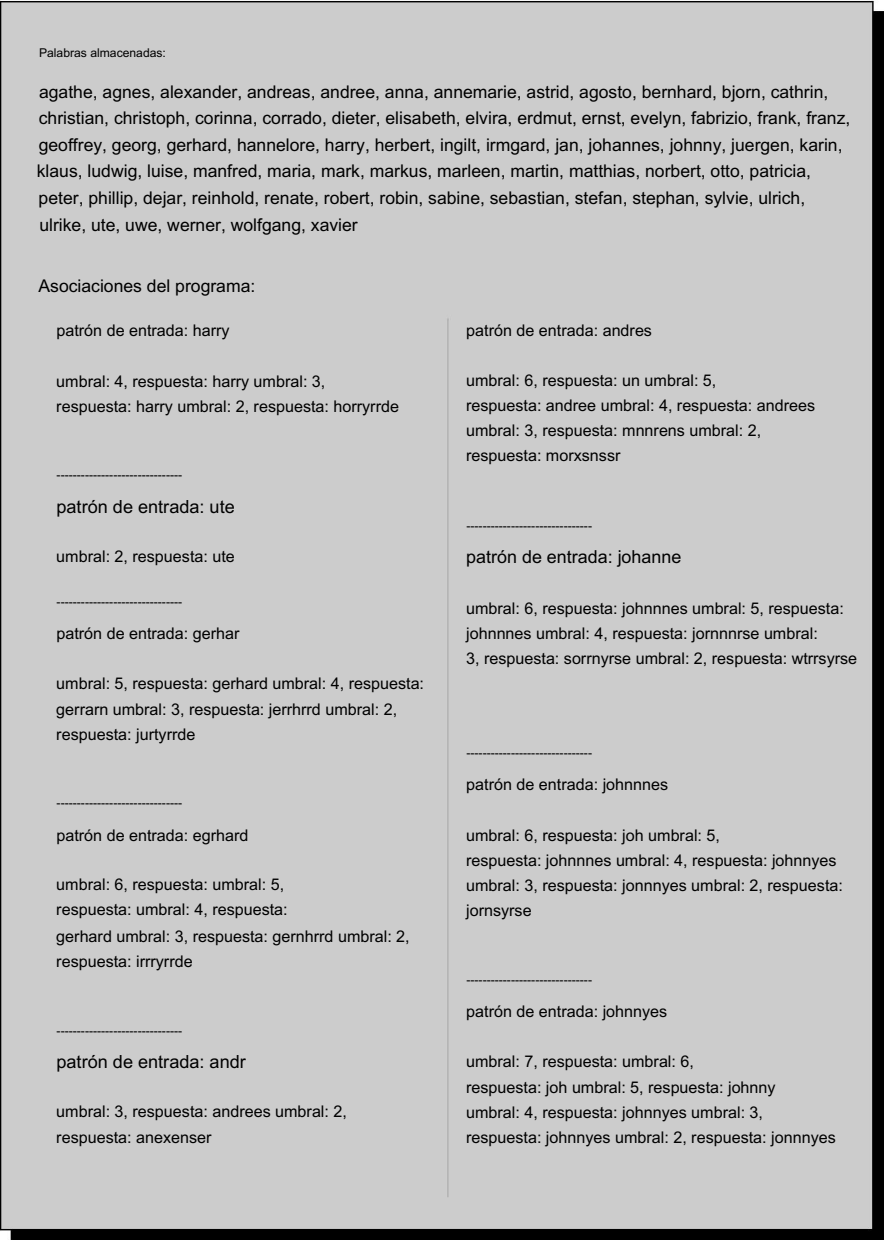


Fig. 9.12 Aplicación del programa de corrección ortográfica a varias entradas aprendidas o erróneas. Las entradas correctas se encuentran con el umbral máximo (es decir, el primer intento). Para entradas erróneas se debe bajar el umbral para una correcta asociación

9.4 Redes lineales con errores mínimos

La regla de Hebb utilizada en los modelos neuronales presentados hasta ahora funciona con asociaciones entre neuronas vecinas. En la memoria asociativa, esto se aprovecha para aprender una asignación de vectores de consulta a objetivos. Esto funciona muy bien en muchos casos, especialmente cuando los vectores de consulta son linealmente independientes. Si esta condición no se cumple, por ejemplo cuando hay demasiados datos de entrenamiento disponibles, surge la pregunta: ¿cómo encontramos la matriz de peso óptima? Óptimo significa que minimiza el error medio.

Los humanos somos capaces de aprender de los errores. La regla de Hebb no ofrece esta posibilidad. El algoritmo de retropropagación, que se describe a continuación, utiliza una solución elegante conocida por la aproximación de funciones para cambiar los pesos de modo que se minimice el error en los datos de entrenamiento.

Sean N pares de vectores de entrenamiento

$$T = \{(q_1, t_1), \dots, (q_N, t_N)\}$$

dado con $q_p \in [0, 1]$, $t_p \in [0, 1]$. Estamos buscando una función $f: [0, 1] \rightarrow [0, 1]$ que minimiza el error cuadrático

$$\sum_{p=1}^N (f(q_p) - t_p)^2$$

sobre los datos. Supongamos primero que los datos no contienen contradicciones. Es decir, no hay un vector de consulta en los datos de entrenamiento que deba asignarse a dos objetivos diferentes. En este caso no es difícil encontrar una función que minimice el error cuadrático. De hecho, existen infinitas funciones que hacen que el error sea cero. Definimos la función

$$f(q) = 0, \text{ si } q \notin \{q_1, \dots, q_N\}$$

y

$$f(q_p) = t_p \quad p \in \{1, \dots, N\}.$$

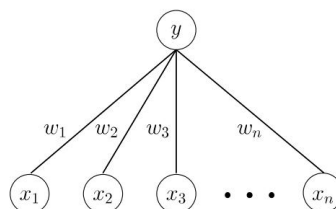
Esta es una función que incluso hace que el error en los datos de entrenamiento sea cero. ¿Qué más podríamos desear? ¿Por qué no estamos contentos con esta función?

La respuesta es: ¡porque queremos construir un sistema inteligente! Inteligente significa, entre otras cosas, que la función aprendida puede generalizarse bien de los datos de entrenamiento a datos nuevos y desconocidos del mismo conjunto de datos representativo. En otras palabras, significa: no queremos un sobreajuste de los datos por memorización. ¿Qué es entonces lo que realmente queremos?

Queremos una función que sea suave y que "iguale" el espacio entre los puntos.

La continuidad y la capacidad de tomar múltiples derivados serían requisitos sensatos. Debido a que incluso con estas condiciones todavía hay infinitas funciones que hacen que el error sea cero, debemos restringir aún más esta clase de funciones.

Fig. 9.13 Una red de dos capas con una salida neurona



9.4.1 Método de los mínimos cuadrados

La opción más simple es un mapeo lineal. Comenzamos con una red de dos capas (Fig. 9.13) en la que la única neurona y de la segunda capa calcula su activación usando

$$y = f \sum_{i=1}^n w_i x_i$$

con $f(x) = x$. El hecho de que aquí solo estemos mirando las neuronas de salida no representa una restricción real porque una red de dos capas con dos o más neuronas de salida siempre se puede separar en redes independientes con neuronas de entrada idénticas para cada una de las neuronas de salida originales. Los pesos de las subredes son todos independientes. El uso de una función sigmoidea en lugar de la activación lineal no brinda ninguna ventaja aquí porque la función sigmoidea aumenta estrictamente de forma monótona y no cambia la relación de orden entre varios valores de salida.

Se desea un vector w que minimice el error cuadrático

$$E(w) = \sum_{p=1}^n (w_p - t_p)^2 = \sum_{p=1}^n \sum_{i=1}^n w_{pi}^2 - 2 \sum_{p=1}^n t_p w_{pi}$$

Como condición necesaria para un mínimo de esta función de error, todas las derivadas parciales deben ser cero. Así requerimos que para $j = 1, \dots, n$:

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^n w_{pj} - 2 \sum_{p=1}^n t_p = 0$$

Multiplicando esto se obtiene

$$\sum_{p=1}^n w_{pj} - \sum_{p=1}^n t_p = 0$$

$$q_{y_0}^{p_{q_0}} q_{y_j}^{p_{q_j}} = t_{p_{q_j}}^{p_{q_j}}$$
$$A_j = \sum_{p,q=1}^n \left(\frac{p}{q} \right)^{\frac{p+q}{2}} \frac{1}{p+q} \left(\frac{p}{q} \right)^{\frac{p+q}{2}} y_{bj} = \sum_{p,q=1}^n \frac{1}{p+q} \left(\frac{p}{q} \right)^{\frac{p+q}{2}} t_{pq} \quad (9.12)$$
$$Aw = b. \quad (9.13)$$

El tiempo de cálculo para configurar la matriz A crece con $O(N \cdot n^2)$ y el tiempo para resolver el sistema de ecuaciones como $O(n^3)$. Este método se puede extender de manera muy simple para incorporar múltiples neuronas de salida porque, como ya se mencionó, para las redes de alimentación directa de dos capas, las neuronas de salida son independientes entre sí.

Como aplicación, ahora determinamos una puntuación lineal para el diagnóstico de apendicitis. A partir de los datos del proyecto LEXMED , que es familiar de la Secc. 8.4.5, usamos el método de mínimos cuadrados para determinar un mapeo lineal de los síntomas a las variables de clase continua AppScore con valores en el intervalo $[0, 1]$ y obtener la combinación lineal

Esta función devuelve variables continuas para AppScore, aunque la variable de clase binaria real App solo toma los valores 0 y 1. Por lo tanto, tenemos que decidir sobre un valor de umbral, como con el perceptrón. El error de clasificación de la puntuación en función del umbral se enumera en la Fig. 9.14 en la página 243 para los datos de entrenamiento y los datos de prueba. Vemos claramente que ambas curvas son casi iguales y tienen

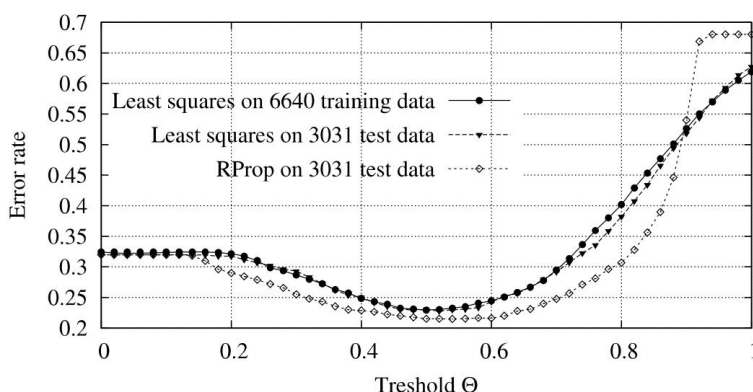


Fig. 9.14 Error de mínimos cuadrados para datos de entrenamiento y prueba

su mínimo en $\Theta = 0.5$. En la pequeña diferencia de las dos curvas vemos que el sobreajuste no es un problema para este método porque el modelo generaliza muy bien a partir de los datos de prueba.

También en la figura está el resultado de la red RProp de tres capas no lineal (Sección 9.5) con un error algo menor para valores de umbral entre 0,2 y 0,9.

Para la aplicación práctica de la puntuación derivada y la determinación correcta del umbral Θ , es importante no solo observar el error, sino también diferenciar por tipo de error (es decir, falso positivo y falso negativo), como se hace en la aplicación LEXMED . en la Fig. 7.10 en la página 142. En la curva ROC que se muestra allí, también se muestra la puntuación calculada aquí. Vemos que el modelo lineal simple es claramente inferior al sistema LEXMED . Evidentemente, las aproximaciones lineales no son lo suficientemente poderosas para muchas aplicaciones complejas.

9.4.3 La regla delta

Los mínimos cuadrados son, como el perceptrón y el aprendizaje del árbol de decisión, un denominado algoritmo de aprendizaje por lotes, a diferencia del aprendizaje incremental. En el aprendizaje por lotes, todos los datos de entrenamiento deben aprenderse en una sola ejecución. Si se agregan nuevos datos de entrenamiento, no se pueden simplemente aprender además de lo que ya existe. Todo el proceso de aprendizaje debe repetirse con el conjunto ampliado. Este problema se resuelve mediante algoritmos de aprendizaje incremental, que pueden adaptar el modelo aprendido a cada nuevo ejemplo adicional. En los algoritmos que veremos en la siguiente discusión, actualizaremos de forma aditiva los pesos para cada nuevo ejemplo de entrenamiento por la regla

$$w_j = w_j + w_j \cdot$$

Para derivar una variante incremental del método de mínimos cuadrados, reconsideramos las n derivadas parciales de la función de error calculadas anteriormente

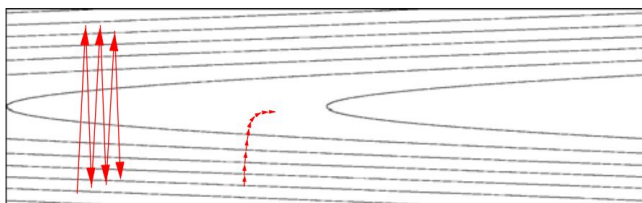


Fig. 9.15 Descenso de gradiente para un η grande (izquierda) y un η muy pequeño (derecha) en un valle que desciende plano hacia la derecha. Para un η grande hay oscilaciones alrededor del valle. Para un η que es demasiado pequeño, en cambio, la convergencia en el valle plano ocurre muy lentamente.

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^P (y_j - t_p) q_j$$

el gradiente

$$\mathbf{E} = \left[\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

como un vector de todas las derivadas parciales de la función de error apunta en la dirección del aumento más fuerte de la función de error en el espacio n -dimensional de los pesos.

Mientras buscamos un mínimo, seguiremos la dirección del gradiente negativo. Como fórmula para cambiar los pesos obtenemos

$$\eta w_j = - \frac{\partial E}{\partial w_j} = - \eta \sum_{p=1}^P (y_j - t_p) q_j$$

donde la tasa de aprendizaje η es una constante positiva libremente seleccionable. Un η más grande acelera la convergencia pero al mismo tiempo aumenta el riesgo de oscilación alrededor de mínimos o valles planos. Por lo tanto, la elección óptima de η no es una tarea sencilla (ver Fig. 9.15). Un η grande, por ejemplo $\eta = 1$, se usa a menudo para empezar y luego se reduce lentamente.

Al reemplazar la activación

$$y_j = \sum_{p=1}^P w_{jp} q_p$$

de la neurona de salida para ejemplo de entrenamiento p , la fórmula se simplifica y aplicado q obtenemos la regla delta

$$w_j = \eta \sum_{p=1}^P (t_p - y_j) q_p$$

Fig. 9.16 Aprendizaje de una red lineal de dos capas con la regla delta. Tenga en cuenta que los cambios de peso siempre ocurren después de aplicar todos los datos de entrenamiento

APRENDIZAJE DELTA (Ejemplos de entrenamiento, η)

Inicializar todos los pesos w_j al azar

Repetir

$w = 0$

Para todos $(q, p, tp) \in \text{TrainingExamples}$

Calcular la salida de la red $y_w = \sum p_j w_j$

$w = w + \eta(tp - y_w)p$

Hasta que w converge

DELTAPRENDIZAJEINCREMENTAL(EjemplosDeEntrenamiento, η)

Inicializar todos los pesos w_j al azar

Repetir

Para todos $(q, p, tp) \in \text{TrainingExamples}$

Calcular la salida de la red $y_w = \sum p_j w_j$

$w = w + \eta(tp - y_w)p$

Hasta que w converge

Fig. 9.17 Variante incremental de la regla delta

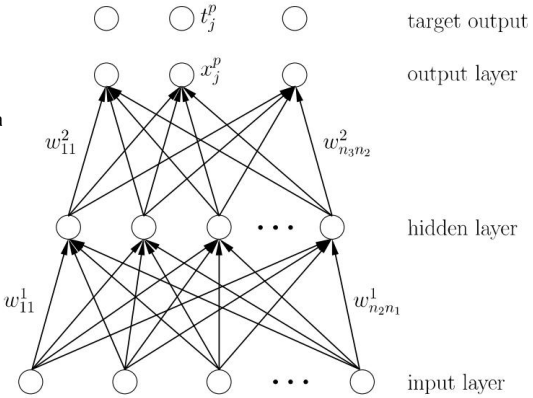
Por lo tanto, para cada ejemplo de entrenamiento, la diferencia entre el objetivo t_p y el p real de la red se calcula para la entrada dada q . Después de sumar la salida y todos los patrones, los pesos se actualizan proporcionalmente a la suma. Este algoritmo se muestra en la figura 9.16.

Vemos que el algoritmo todavía no es realmente incremental porque los cambios de peso solo ocurren después de que todos los ejemplos de entrenamiento se hayan aplicado una vez. Podemos corregir esta deficiencia cambiando directamente los pesos (descenso de gradiente incremental) después de cada ejemplo de entrenamiento (Fig. 9.17), lo que, estrictamente hablando, ya no es una implementación correcta de la regla delta.

9.4.4 Comparación con el Perceptron

La regla de aprendizaje para perceptrones presentada en la Secc. 8.2.1, el método de los mínimos cuadrados y la regla delta se pueden utilizar para generar funciones lineales a partir de datos. Sin embargo, para los perceptrones, a diferencia de los otros métodos, se aprende un clasificador para clases linealmente separables a través de la decisión del umbral. Los otros dos métodos, sin embargo, generan una aproximación lineal a los datos. Como se muestra en la Secc. 9.4.2, se puede generar un clasificador a partir del mapeo lineal, si se desea, mediante la aplicación de una función de umbral.

Fig. 9.18 Una red de retropropagación de tres capas con n_1 neuronas en la primera, n_2 neuronas en la segunda y n_3 neuronas en la tercera capa



El perceptrón y la regla delta son algoritmos iterativos para los cuales el tiempo hasta la convergencia depende en gran medida de los datos. En el caso de datos linealmente separables, se puede encontrar un límite superior en el número de pasos de iteración para el perceptrón. Para la regla delta, en cambio, solo hay una garantía de convergencia asintótica sin límite [HKP91].

Para los mínimos cuadrados, el aprendizaje consiste en establecer y resolver un sistema lineal de ecuaciones para el vector de peso. Por lo tanto, existe un límite estricto en el tiempo de cálculo. Debido a esto, el método de los mínimos cuadrados siempre es preferible cuando no se necesita un aprendizaje incremental.

9.5 El algoritmo de retropropagación

Con el algoritmo de retropropagación, ahora presentamos el modelo neuronal más utilizado. La razón de su uso generalizado es su versatilidad universal para tareas de aproximación arbitraria. El algoritmo se origina directamente de la regla delta incremental. A diferencia de la regla delta, aplica una función sigmoidea no lineal sobre la suma ponderada de las entradas como su función de activación. Además, una red de retropropagación puede tener más de dos capas de neuronas. El algoritmo se dio a conocer a través del artículo [RHR86] en la legendaria colección PDP [RM86].

En la Fig. 9.18, una red de retropropagación típica con una capa de entrada, una capa oculta, y se muestra una capa de salida. Dado que el valor de salida actual x_j de la salida de la capa se compara con el valor de salida objetivo t_j , éstos se dibujan paralelos a j , entre sí. Aparte de las neuronas de entrada, todas las neuronas calculan su valor actual x_j por la regla

$$x_j = f \left(\sum_i w_{ji} x_i \right)$$

(9.14)

donde n es el número de neuronas en la capa anterior. Usamos la función sigmoide

$$f(x) = \frac{1}{1 + e^{-x}}.$$

De manera análoga a la regla delta incremental, los pesos cambian proporcionalmente al gradiente negativo de la función de error cuadrático sumado sobre las neuronas de salida

$$E_p(w) = \frac{1}{2} \sum_{k \text{ salida}} (t_k^{\text{pags}} - x_k)^2$$

para el patrón de entrenamiento p :

$$\Delta w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

Para derivar la regla de aprendizaje, la expresión anterior se sustituye por E_p . Dentro de la expresión, x_k se reemplaza por (9.14) en la página 246. Dentro de la ecuación, las salidas x_i de las neuronas de la siguiente capa más profunda ocurren recursivamente, etc. Mediante múltiples aplicaciones de la regla de la cadena (ver [RHR86] o [Zel94]) obtenemos la regla de aprendizaje backpropagation

$$\Delta w_{ji} = \eta \delta_j x_{pi}$$

con

$$\delta_j = \sum_{k \text{ salida}} (t_k^{\text{pags}} - x_k) (t_k^{\text{pags}} - x_k) \text{ si } j \text{ es una neurona de salida,}$$

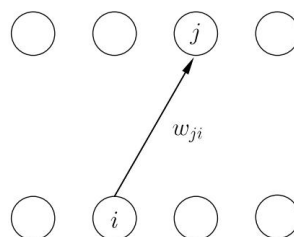
$$\delta_j = \sum_{k \text{ oculta}} \delta_k w_{kj} (1 - x_j) \text{ si } j \text{ es una neurona oculta,}$$

que también se denota la regla delta generalizada. Para todas las neuronas, la fórmula para cambiar el peso w_{ji} de la neurona i a la neurona j (ver Fig. 9.19 en la página 248) contiene, como la regla de Hebb, un término $x_i x_j$. El nuevo factor $(1 - x_j \delta_j)$ crea el simme que falta en la regla de Hebb, entre las activaciones 0 y 1 de neurona j . Para las neuronas de salida, el factor $(t_j^{\text{pags}} - x_j)$ se encarga de un cambio de peso proporcional al error δ_j . Para las neuronas ocultas, el valor δ_j recursivamente de todos los δ_k de neurona k se calcula. La ejecución completa de las neuronas del siguiente nivel superior.

del proceso de aprendizaje se muestra en la Fig. 9.20 en la página 248.

Después de calcular la salida de la red (propagación directa) para un ejemplo de entrenamiento, se calcula el error de aproximación. Esto luego se usa durante la propagación hacia atrás para alterar los pesos hacia atrás de una capa a otra. Luego, todo el proceso se aplica a todos los ejemplos de entrenamiento y se repite hasta que los pesos ya no cambian o se alcanza un límite de tiempo.

Fig. 9.19 Designación de las neuronas y pesos para la aplicación de la regla de retropropagación



PROPAGACIÓN HACIA ATRÁS (Ejemplos de entrenamiento, η)

Inicializar todos los pesos w_j a valores aleatorios

Repetir

Para todos (q, p, t, p) TrainingExamples p a

Aplicar el vector de consulta q 2. la capa de entrada 1.

Propagación directa: Para todas

las capas desde la primera capa oculta hacia arriba Para cada neurona de la capa

Calcular activación $x_j = f(\sum_{i=1}^n w_{ji} x_i)$

3. Cálculo del error cuadrático $E_p(w)$

4. Propagación hacia atrás: Para

todos los niveles de pesos desde el último hacia abajo Para

cada peso $w_{ji} = w_{ji} + \eta \delta_j x_{ip}$

w_{ji} δ_j x_{ip}

Hasta que w converge o se alcanza el límite de tiempo

Fig. 9.20 El algoritmo de retropropagación

Si construimos una red con al menos una capa oculta, se pueden aprender mapeos no lineales. Sin capas ocultas, las neuronas de salida no son más poderosas que una neurona lineal, a pesar de la función sigmoidea. La razón de esto es que la función sigmoidea es estrictamente monótona. Lo mismo ocurre con las redes multicapa que solo utilizan una función lineal como función de activación, por ejemplo, la función de identidad.

Esto se debe a que las ejecuciones encadenadas de asignaciones lineales son lineales en conjunto.

Al igual que con el perceptrón, la clase de las funciones que se pueden representar también se agrandan si usamos una función sigmoidea variable

$$f(x) = \frac{1}{1 + e^{-(x-\Theta)}}$$

con umbral Θ . Esto se implementa de forma análoga a la que se muestra en la Secc. 8.2, en el que se inserta en la capa de entrada y en cada capa oculta una neurona cuya activación siempre tiene el valor uno y que está conectada a las neuronas del siguiente nivel superior. Los pesos de estas conexiones se aprenden normalmente y representan el umbral Θ de las neuronas sucesoras.

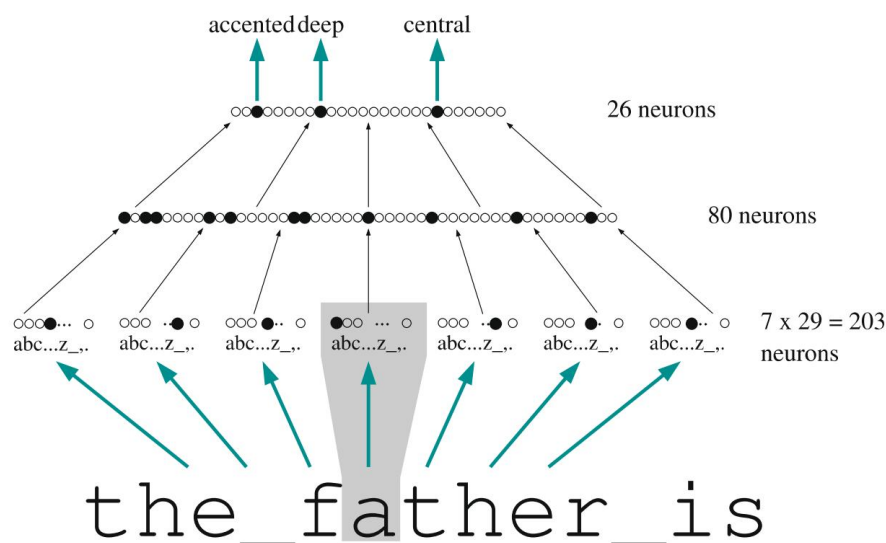


Fig. 9.21 La red NETtalk asigna un texto a sus atributos de pronunciación

9.5.1 NETtalk: una red aprende a hablar

Sejnowski y Rosenberg demostraron de manera muy impresionante en 1986 lo que la retropropagación es capaz de realizar [SR86]. Construyeron un sistema que es capaz de leer texto en inglés en voz alta desde un archivo de texto. La arquitectura de la red que se muestra en la figura 9.21 consta de una capa de entrada con $7 \times 29 = 203$ neuronas en el que se codifican la letra actual y las tres letras anteriores, así como las tres letras posteriores. Para cada una de estas siete letras, se reservan 29 neuronas para el caracteres "a...z ,.". La entrada se asigna a las 26 neuronas de salida sobre 80 neuronas ocultas, cada una de las cuales representa un fonema específico. por ejemplo, el La "a" en "padre" se pronuncia profunda, acentuada y central. La red fue entrenada con 1.000 palabras, que fueron aplicadas aleatoriamente una tras otra letra por letra. Para cada letra, la salida objetivo se proporcionó manualmente para su entonación. Para traducir los atributos de entonación en sonidos reales, parte del sistema de síntesis del habla Se utilizó DECtalk. A través de la interconectividad completa, la red contiene un total de $203 \times 80 + 80 \times 26 = 18320$ pesos.

El sistema fue entrenado usando un simulador en un VAX 780 con alrededor de 50 ciclos. sobre todas las palabras. Así, con unos 5 caracteres por palabra de media, unos $5 \cdot 50 \cdot 1000 =$ Se necesitaron 250 000 iteraciones del algoritmo de retropropagación. A un ritmo de aproximadamente un carácter por segundo, esto significa aproximadamente 69 horas de tiempo de cálculo. El los desarrolladores observaron muchas propiedades del sistema que son bastante similares a las de los humanos aprendiendo. Al principio, el sistema solo puede hablar de forma poco clara o utilizar palabras sencillas. Con tiempo continuó mejorando y finalmente alcanzó el 95% de corrección de las letras pronunciadas.

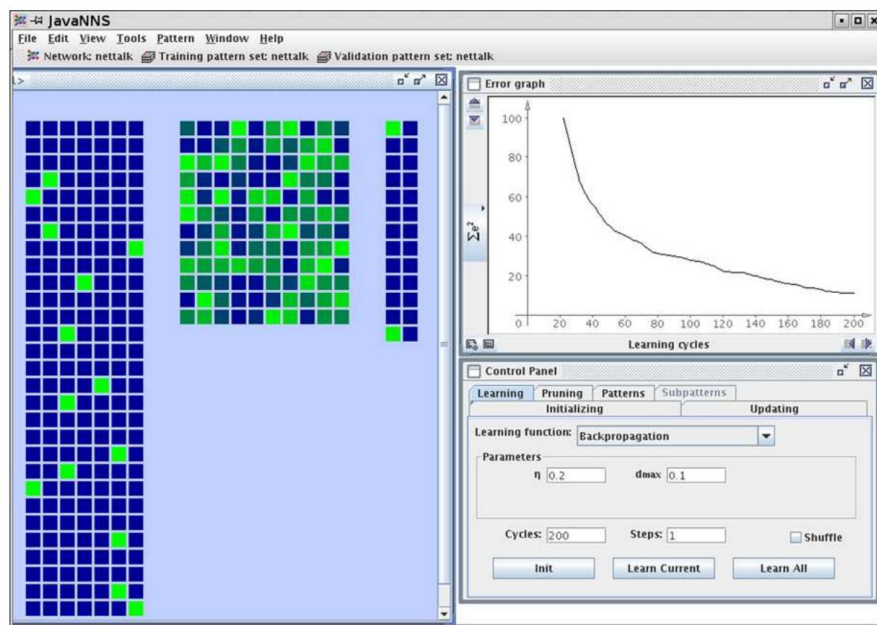


Fig. 9.22 La red NETtalk en JNNS. En la ventana izquierda de izquierda a derecha vemos las $7 \cdot 29$ neuronas de entrada, 80 ocultas y 26 neuronas de salida. Debido a su gran número, los pesos se omiten de la red. Arriba a la derecha hay una curva de aprendizaje que muestra el desarrollo del error cuadrático a lo largo del tiempo.

Para experimentos visuales con redes neuronales, se recomienda Java Neural Network Simulator JNNS [Zel94]. La red NETtalk, cargada y entrenada con JNNS, se muestra en la Fig. 9.22.

9.5.2 Aprendizaje de heurísticas para probadores de teoremas

En el cap. 6, se discutieron algoritmos para la búsqueda heurística, tales como el algoritmo A y el algoritmo IDA. Para alcanzar una reducción significativa del espacio de búsqueda, se necesita una buena heurística para la implementación de estos algoritmos. Insecto. 4.1 Se demostró el problema de la explosión del espacio de búsqueda durante la búsqueda de una demostración por parte de los probadores de teoremas. Este problema es causado por la gran cantidad de posibles pasos de inferencia en cada paso.

Ahora intentamos construir módulos de control de pruebas heurísticas que evalúen las diversas alternativas para el siguiente paso y luego elijan la alternativa con la mejor calificación. En el caso de resolución, la calificación de las cláusulas disponibles podría ser realizada por una función que, en base a ciertos atributos de las cláusulas como el número de literales, la complejidad de los términos, etc., calcule un valor para cada par de cláusulas resol. cláusulas capaces. En [ESS89, SE90] se aprendió tal heurística para el probador de teoremas SETHEO usando retropropagación. Mientras aprende la heurística, los atributos de la

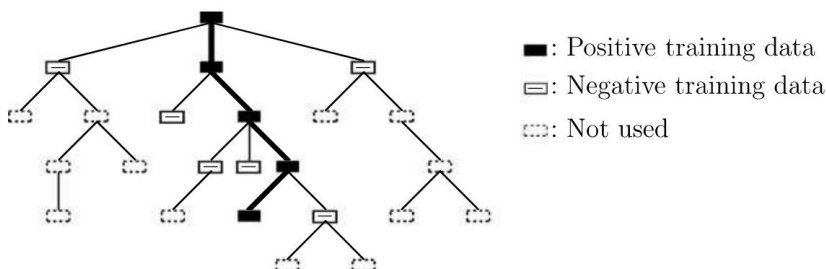


Fig. 9.23 Árbol de búsqueda con una ruta exitosa cuyos nodos se evalúan como positivos y las ramas negativas calificadas como no exitosas

la cláusula actual se guardó como datos de entrenamiento para la clase positiva para cada prueba encontrada en cada sucursal durante la búsqueda. En todas las ramas que no condujeron a una prueba, el los atributos de la cláusula actual se guardaron como datos de entrenamiento para la clase negativa. Tal árbol, con una ruta exitosa y las clasificaciones de nodos correspondientes, se dibuja en la figura 9.23.

Se entrenó una red de retropropagación con estos datos y luego se utilizó para evaluar cláusulas en el prover. Para cada cláusula, se calcularon, normalizaron y codificaron 16 atributos numéricos en una neurona de entrada. La red fue entrenada con 25 ocultos neuronas y una neurona de salida para la clase (positiva/negativa).

Se demostró que el número de intentos de inferencia se puede reducir en muchos órdenes de magnitud para problemas difíciles, lo que finalmente reduce el tiempo de cálculo de horas a segundos. De este modo se hizo posible probar teoremas que, sin heurísticas, estaban fuera de alcance.

9.5.3 Problemas y mejoras

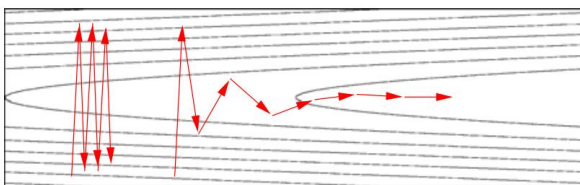
Backpropagation tiene ahora 25 años y ha demostrado su valía en varias aplicaciones, por ejemplo en reconocimiento de patrones y en robótica. Sin embargo, su aplicación es a veces problemática. Especialmente cuando la red tiene muchos miles de pesos y hay muchos datos de entrenamiento para aprender, surgen dos problemas:

La red a menudo converge a los mínimos locales de la función de error. Además, la retropropagación a menudo converge muy lentamente. Esto significa que muchas iteraciones sobre todos los patrones de entrenamiento son necesarios. Se han sugerido muchas mejoras para aliviar estos problemas. Como se menciona en la Secc. 9.4.3, las oscilaciones pueden evitarse reduciendo lentamente la tasa de aprendizaje η como se muestra en la Fig. 9.15 en la página 244.

Otro método para reducir las oscilaciones es el uso de un término de cantidad de movimiento mientras actualizando los pesos, lo que asegura que la dirección de descenso del gradiente no cambiar demasiado dramáticamente de un paso al siguiente. Aquí para el peso actual cambiar $pwj_i(t)$ en el tiempo t se agrega otra parte del cambio $pwj_i(t-1)$ del paso anterior. La regla de aprendizaje luego cambia a

$$pwj_i(t) = \eta \delta p_j x_i^{pq} + \gamma pwj_i(t-1)$$

Fig. 9.24 Dirección abrupta
los cambios son suavizados por
el uso del impulso
término. Una iteración sin el
término de impulso (izquierda) en
comparación con la iteración con
el término de impulso (derecha)



con un parámetro y entre cero y uno. Esto se representa en dos dimensiones.

ejemplo en la figura 9.24.

Otra idea es minimizar la función de error lineal en lugar del error cuadrático.

función, que reduce el problema de la convergencia lenta en valles planos.

El descenso de gradiente en la retropropagación se basa en última instancia en una aproximación lineal de la función de error. Quickprop, un algoritmo que utiliza una aproximación cuadrática a la función de error y por lo tanto logra una convergencia más rápida, fue creado por Scott Fahlman.

A través de la unificación inteligente de las mejoras mencionadas y otros trucos heurísticos, Martin Riedmiller logró una mayor optimización con el algoritmo RProp [RB93]. RProp ha reemplazado a la retropropagación y es el nuevo estado de la
Algoritmo de aproximación de red neuronal feedforward de arte. Insecto. 8.8 aplicamos RProp a la clasificación de los datos de apendicitis y logró un error que es aproximadamente del mismo tamaño que el de un árbol de decisión aprendido.

9.6 Máquinas de vectores de soporte

Las redes neuronales feedforward con solo una capa de pesos son lineales. linealidad conduce a redes simples y de rápido aprendizaje con convergencia garantizada. Además, el peligro de sobreajuste es pequeño para los modelos lineales. Para muchas aplicaciones, sin embargo, los modelos lineales no son lo suficientemente fuertes, por ejemplo porque las clases relevantes no son linealmente separables. Aquí entran en uso las redes multicapa como la retropropagación, con la consecuencia de que los mínimos locales, la convergencia problemas y puede ocurrir un sobreajuste.

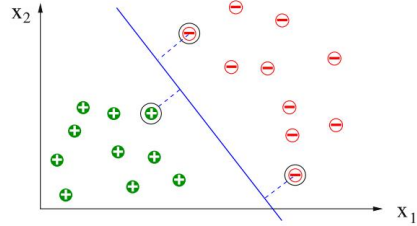
Un enfoque prometedor, que reúne las ventajas de los modelos lineales y no lineales, sigue la teoría de las máquinas de vectores de soporte (SVM), que veremos a continuación.
esbozar aproximadamente usando un problema de dos clases.⁴

En el caso de dos clases linealmente separables, es fácil encontrar un hiper divisorio plano, por ejemplo con la regla de aprendizaje del perceptrón. Sin embargo, suele haber infinitos de tales planos, como en el ejemplo bidimensional de la figura 9.25 en página 253. Estamos buscando un avión que tenga la mayor distancia mínima a ambos clases Este plano suele estar definido únicamente por unos pocos puntos en el área del borde. Estos puntos, los llamados vectores de apoyo, tienen todos la misma distancia a la división

⁴Las máquinas de vectores de soporte no son redes neuronales. Sin embargo, debido a su desarrollo histórico y su relación matemática con las redes lineales, se analizan aquí.

Fig. 9.25 Dos clases con la línea divisoria máxima.

Los puntos encerrados en un círculo son los vectores de soporte.



línea. Para encontrar los vectores de soporte, existe un algoritmo de optimización eficiente. Es interesante que el hiperplano de división óptimo esté determinado por algunos parámetros, a saber, por los vectores de soporte. Por lo tanto, el peligro de sobreajuste es pequeño.

Las máquinas de vectores de soporte aplican este algoritmo a problemas separables no linealmente en un proceso de dos pasos: en el primer paso, se aplica una transformación no lineal a los datos, con la propiedad de que los datos transformados son separables linealmente. En el segundo paso se determinan los vectores de soporte en el espacio transformado.

El primer paso es muy interesante, pero no del todo sencillo. De hecho, siempre es posible hacer que las clases sean linealmente separables transformando el espacio vectorial, siempre que los datos no contengan contradicciones.⁵ Tal separación puede lograrse, por ejemplo, introduciendo una nueva dimensión ($n + 1$) y la definición

$$x_{n+1} = \begin{cases} 1 & \text{si } x \text{ clase 1,} \\ x & \text{clase 0.} \end{cases}$$

Sin embargo, esta fórmula no ayuda mucho porque no es aplicable a nuevos puntos de una clase desconocida que se van a clasificar. Por lo tanto, necesitamos una transformación general que sea lo más independiente posible de los datos actuales. Se puede demostrar que existen tales transformaciones genéricas incluso para límites de división de clases formados arbitrariamente en el espacio vectorial original. En el espacio transformado, los datos son linealmente separables. Sin embargo, el número de dimensiones del nuevo espacio vectorial crece exponencialmente con el número de dimensiones del espacio vectorial original.

Sin embargo, la gran cantidad de nuevas dimensiones no es tan problemática porque, cuando se usan vectores de soporte, el plano divisorio, como se mencionó anteriormente, está determinado por solo unos pocos parámetros.

La transformación no lineal central del espacio vectorial se denomina kernel, por lo que las máquinas de vectores de soporte también se conocen como métodos kernel. La teoría SVM original desarrollada para tareas de clasificación se ha ampliado y ahora también se puede utilizar en problemas de regresión.

Las matemáticas utilizadas aquí son muy interesantes, pero demasiado extensas para una introducción inicial. Para profundizar en esta rama joven y prometedora del aprendizaje automático, remitimos al lector a [SS02, Alp04] y [Bur98].

⁵Un punto de datos es contradictorio si pertenece a ambas clases.

9.7 Aplicaciones

Además de los ejemplos indicados aquí, existen innumerables aplicaciones para redes neuronales en todas las ramas de la industria. Un área muy importante es el reconocimiento de patrones en todas sus formas, ya sea análisis de fotografías para reconocer personas o rostros, reconocimiento de bancos de peces a partir de imágenes de sonar, reconocimiento y clasificación de vehículos militares a partir de escaneos de radar, y muchos más. Pero las redes neuronales también se pueden entrenar para reconocer el lenguaje hablado y el texto escrito a mano.

Las redes neuronales no solo se utilizan para reconocer objetos y escenas. También están capacitados para controlar robots simples utilizando datos de sensores, así como para controlar heurísticamente la búsqueda en computadoras de backgammon y ajedrez. Particularmente interesantes en los juegos son las técnicas de aprendizaje por refuerzo Secc. 10.8, por ejemplo con redes neuronales.

Durante mucho tiempo, las redes neuronales, además de los métodos estadísticos, se han utilizado con éxito para pronosticar los mercados bursátiles y evaluar la solvencia de los clientes bancarios.

Para muchas de estas aplicaciones, también se pueden usar otros algoritmos de aprendizaje automático. Sin embargo, hasta la fecha, las redes neuronales han sido más exitosas y populares que todos los demás algoritmos de aprendizaje automático. Sin embargo, el predominio histórico de las redes neuronales está retrocediendo lentamente a favor de otros métodos debido al gran éxito comercial de la minería de datos y las máquinas de vectores de soporte.

9.8 Resumen y perspectiva

Con el perceptrón, la regla delta y la retropropagación, hemos introducido la clase más importante de redes neuronales y su relación con las puntuaciones y el ingenio Bayes por un lado, pero también con el método de los mínimos cuadrados. A continuación vimos las fascinantes redes de Hopfield, inspiradas en modelos biológicos, que son, sin embargo, difíciles de manejar en la práctica debido a su compleja dinámica. De mayor importancia en la práctica son los diversos modelos de memoria asociativa.

En todos los modelos neuronales, la información se almacena distribuida en muchos pesos. Debido a esto, la muerte de algunas neuronas en el cerebro no tiene un efecto notable en la función del cerebro. A través de su almacenamiento distribuido de datos, la red es robusta frente a pequeñas interrupciones. Esto también está relacionado con la capacidad de reconocer patrones con errores que se muestran en múltiples ejemplos.

Sin embargo, la representación distribuida del conocimiento tiene la desventaja de que es difícil para el ingeniero del conocimiento localizar la información. Es prácticamente imposible analizar y comprender los muchos pesos en una red neuronal completamente entrenada. Para un árbol de decisión aprendido, en cambio, es fácil comprender el conocimiento aprendido e incluso representarlo como una fórmula lógica. La lógica de predicados, que permite formalizar relaciones, es especialmente expresiva y elegante. Por ejemplo, el predicado abuela (karen, clyde) es fácil de entender. Este tipo de relación también se puede aprender con redes neuronales, pero ni siquiera es posible localizar la "neurona abuela" en la red. Por lo tanto, todavía hay problemas para conectar redes neuronales con sistemas de procesamiento de símbolos.

De la gran cantidad de modelos neuronales interesantes, muchos no podrían tratarse aquí. Por ejemplo, los mapas de autoorganización introducidos por Kohonen son un mapeo motivado biológicamente desde una capa sensora de neuronas a una segunda capa de neuronas con la propiedad de que este mapeo es adaptativo y conserva la similitud.

Un problema con las redes presentadas aquí surge durante el aprendizaje incremental. Si, por ejemplo, una red de retropropagación completamente entrenada se entrena aún más con nuevos patrones, entonces muchos (y potencialmente todos) de los patrones antiguos se olvidan rápidamente. Para resolver este problema, Carpenter y Grossberg desarrollaron la teoría de la resonancia adaptativa (ART), que resultó en toda una línea de modelos neuronales.

Como literatura adicional recomendamos los libros de texto [Bis05, RMS92, Roj96, Zel94]. Los interesados en leer la obra original más importante en este apasionante campo pueden consultar los dos volúmenes recopilados [AR88, APR90].

9.9 Ejercicios

9.9.1 De la biología a la simulación

Ejercicio 9.1 Muestre la simetría puntual de la función sigmoidea a $(\theta, \frac{1}{2})$.

9.9.2 Redes Hopfield

Ejercicio 9.2 Use el applet de la red Hopfield en <http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html> y pruebe la capacidad de la memoria para patrones correlacionados y no correlacionados (con bits establecidos al azar) con un 10×10 tamaño de la cuadrícula. Use un patrón con un 10 % de ruido para las pruebas.

Ejercicio 9.3 Compare el límite teórico $N = 0.146 n$ para el número máximo de patrones que se pueden almacenar, dado en la Secc. 9.2.2, con capacidad de almacenamiento binario clásico del mismo tamaño.

9.9.3 Redes lineales con errores mínimos

Ejercicio 9.4

- (a) Escriba un programa para aprender un mapeo lineal con el método de mínimos cuadrados medios. Esto es bastante simple: solo se deben establecer las ecuaciones normales por (9.12) en la página 242 y luego resolver el sistema de ecuaciones.
- (a) Aplique este programa a los datos de apendicitis en el sitio web de este libro y determine una puntuación lineal. Dé el error como una función del umbral, como en la figura 9.14 en la página 243. (c) Ahora determine la curva ROC para esta puntuación.

9.9.4 Propagación hacia atrás

Ejercicio 9.5 Utilizando un programa de retropropagación (por ejemplo, en JNNS o KNIME), cree una red con ocho neuronas de entrada, ocho neuronas de salida y tres neuronas ocultas. Luego entrene la red con ocho pares de datos de entrenamiento q $erty$ que, en el pp con el puntal $, q$ pth vector q el pth bit es uno y todos los demás bits son cero. La red tiene así la tarea simple de aprender un mapeo idéntico. Tal red se llama codificador 8-3-8. Después del proceso de aprendizaje, observe la codificación de las tres neuronas ocultas para la entrada de los ocho vectores de entrada aprendidos. ¿Que notaste? Ahora repite el experimento, reduce el número de neuronas ocultas a dos y luego a una.

Ejercicio 9.6 Para mostrar que las redes de retropropagación pueden dividir conjuntos separables no linealmente, entrene la función XOR. Los datos de entrenamiento para esto son: $((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 0)$. (a) Cree una red con dos neuronas cada una en la capa de entrada y la capa oculta, y una neurona en la capa de salida, y entrene esta red. (b) Ahora elimine las neuronas ocultas y conecte la capa de entrada directamente a las neuronas de salida. ¿Qué observas?

Ejercicio 9.7

(a) Muestre que cualquier red de retropropagación de múltiples capas con una función de activación lineal es tan poderosa como una de dos capas. Para esto es suficiente mostrar que ejecuciones sucesivas de mapeos lineales es un mapeo lineal. (b) Muestre que una red de retropropagación de dos capas con cualquier función de activación estrictamente monótona no es más poderosa para las tareas de clasificación que una sin función de activación o con una función de activación lineal.

9.9.5 Máquinas de vectores de soporte

Ejercicio 9.8 Dos conjuntos bidimensionales no linealmente separables de datos de entrenamiento M^+ y M^- están dados. Todos los puntos en M^+ están dentro del círculo unitario $x^2 + y^2 = 1$ y todos los puntos en M^- están afuera. Dé una transformación de coordenadas $f: \mathbb{R}^2 \rightarrow \mathbb{R}^n$ que haga que los datos sean linealmente separables. Da la ecuación de la línea divisoria y dibuja los dos espacios y los puntos de datos.

10.1 Introducción

Todos los algoritmos de aprendizaje descritos hasta ahora, excepto los algoritmos de agrupamiento, pertenecen a la clase de aprendizaje supervisado. En el aprendizaje supervisado, se supone que el agente aprende un mapeo de las variables de entrada a las variables de salida. Aquí es importante que para cada ejemplo de entrenamiento individual, se proporcionen todos los valores tanto para las variables de entrada como para las variables de salida. En otras palabras, necesitamos un maestro o una base de datos en la que el mapeo a aprender esté definido aproximadamente para un número suficiente de valores de entrada. La única tarea del algoritmo de aprendizaje automático es filtrar el ruido de los datos y encontrar una función que se aproxime bien al mapeo, incluso entre los puntos de datos dados.

En el aprendizaje por refuerzo la situación es diferente y más difícil porque no hay datos de entrenamiento disponibles. Comenzamos con un ejemplo ilustrativo simple de la robótica, que luego se utiliza como una aplicación para los distintos algoritmos.

El aprendizaje por refuerzo es muy valioso en el campo de la robótica, donde las tareas a realizar son lo suficientemente complejas como para desafiar la codificación como programas y no hay datos de entrenamiento disponibles. La tarea del robot consiste en averiguar, mediante prueba y error (o éxito), qué acciones son buenas en una determinada situación y cuáles no. En muchos casos, los humanos aprendemos de una manera muy similar. Por ejemplo, cuando un niño aprende a caminar, esto suele suceder sin instrucción, sino simplemente mediante el refuerzo. Los intentos exitosos de caminar son recompensados con el progreso hacia adelante, y los intentos fallidos son penalizados con caídas a menudo dolorosas. El refuerzo positivo y negativo también son factores importantes en el aprendizaje exitoso en la escuela y en muchos deportes (ver Fig. 10.1 en la página 258).

Una tarea de movimiento muy simplificada se aprende en el siguiente ejemplo.

Ejemplo 10.1 Un robot cuyo mecanismo consta solo de un bloque rectangular y un brazo con dos articulaciones g_y y g_x se muestra en la Fig. 10.2 en la página 258 (ver [KMK97]).

Las únicas acciones posibles del robot son el movimiento de g_y hacia arriba o hacia abajo y de g_x hacia la derecha o hacia la izquierda. Además, solo permitimos movimientos de unidades discretas fijas (por ejemplo, de incrementos de 10 grados). La tarea del agente consiste en aprender una política



Fig. 10.1 “¿Tal vez la próxima vez debería comenzar a girar un poco antes o ir más lento?”—Aprendizaje del refuerzo negativo

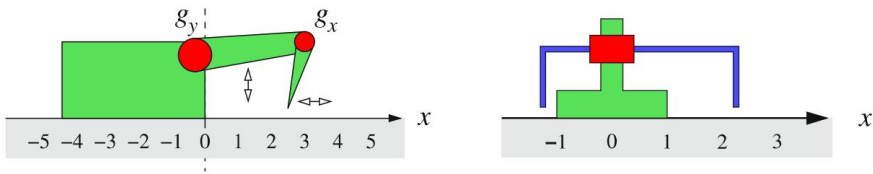


Fig. 10.2 Al mover ambas articulaciones, el robot que se arrastra a la izquierda de la imagen puede moverse hacia adelante y hacia atrás. El robot que camina en el lado derecho debe mover el marco de manera correspondiente hacia arriba y hacia abajo o hacia la izquierda y hacia la derecha. La retroalimentación del movimiento del robot es positiva para el movimiento hacia la derecha y negativa para el movimiento hacia la izquierda.

lo que le permite moverse lo más rápido posible hacia la derecha. El robot andante de la figura 10.2 funciona de manera análoga dentro del mismo espacio de estado bidimensional.

Una secuencia de acción exitosa se muestra en la Tabla 10.1 en la página 259. La acción en el tiempo $t = 2$ da como resultado que el brazo cargado mueva el cuerpo una unidad de longitud hacia la derecha. Buenas animaciones de este ejemplo se pueden encontrar a través de [KMK97] y [Tok06].

Antes de entrar en el algoritmo de aprendizaje, debemos modelar adecuadamente la tarea matemáticamente. Describimos el estado del robot por las dos variables g_x y g_y para la posición de las articulaciones, cada una con un número finito de valores discretos. El estado del robot se codifica así como un vector (g_x, g_y) . El número de posiciones conjuntas posibles es n_x , respectivamente n_y . Usamos la posición horizontal del cuerpo del robot, que puede tomar valores reales, para evaluar las acciones del robot. Los movimientos a la derecha son recompensados

Tabla 10.1 Ciclo de una serie periódica de movimientos con avance sistemático

Robot rastrero	robot corriendo	Tiempo t	Estado		Acción de recompensa	
			gy	gx	x	en
		0	Arriba	Izquierda	0	Bien
		1	Arriba	Bien	0	Abajo
		2	Abajo a la derecha		0	Izquierda
		3	Abajo Izquierda1			Arriba

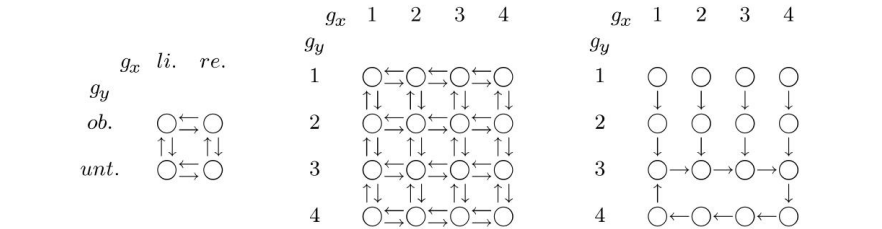


Fig. 10.3 El espacio de estado del robot de ejemplo en el caso de dos posibles posiciones para cada uno de las articulaciones (izquierda) y en el caso de cuatro posiciones horizontales y verticales para cada uno (centro). En la imagen correcta se da una política óptima

con cambios positivos a x, y los movimientos a la izquierda son castigados con negativos cambios.

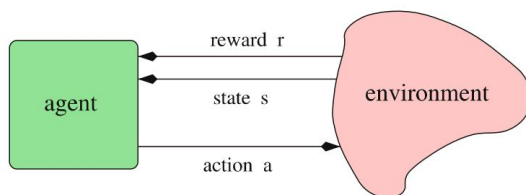
En la figura 10.3, el espacio de estado para dos variantes de esto se muestra en forma simplificada.¹ En el ejemplo de la izquierda, ambas articulaciones tienen dos posiciones, y en el ejemplo del medio cada uno tiene cuatro posiciones. Una política óptima se da en la Fig. 10.3 a la derecha.

10.2 La tarea

Como se muestra en la figura 10.4 en la página 260, distinguimos entre el agente y su entorno. En el momento t el mundo, que incluye al agente y su entorno, es descrito por un estado s_t . El conjunto S es una abstracción de los posibles estados reales del mundo porque, por un lado, el mundo no se puede describir exactamente, y por otro lado, el agente a menudo solo tiene información incompleta sobre el

¹El espacio de movimiento del brazo que consta de arcos se representa como una cuadrícula en ángulo recto.

Fig. 10.4 El agente y su interacción con el entorno



estado debido a errores de medición. Entonces, el agente realiza una acción en A en el tiempo t . Esta acción cambia el mundo y por lo tanto da como resultado el estado s_{t+1} en el tiempo $t+1$.

La función de transición de estado δ definida por el entorno determina el nuevo estado $s_{t+1} = \delta(s_t, a_t)$.

No puede ser influenciado por el agente.

Después de ejecutar la acción en A , el agente obtiene una recompensa inmediata $r_t = r(s_t, a_t)$ (ver Fig. 10.4). La recompensa inmediata $r_t = r(s_t, a_t)$ siempre depende del estado actual y de la acción ejecutada. $r(s_t, a_t) = 0$ significa que el agente no recibe retroalimentación inmediata por la acción en A . Durante el aprendizaje, $r_t > 0$ debería resultar en un refuerzo positivo y $r_t < 0$ en un refuerzo negativo de la evaluación de la acción en el estado s_t .

Especialmente en el aprendizaje por refuerzo, se están estudiando aplicaciones en las que no se produce una recompensa inmediata durante mucho tiempo. Un jugador de ajedrez, por ejemplo, aprende a mejorar su juego a partir de partidas ganadas o perdidas, incluso si no obtiene una recompensa inmediata por todos los movimientos individuales. Aquí podemos ver la dificultad de asignar la recompensa al final de una secuencia de acciones a todas las acciones de la secuencia que condujeron a este punto (problema de asignación de créditos).

En el caso del robot que se arrastra, el estado consiste en la posición de las dos articulaciones, es decir, $s = (g_x, g_y)$. La recompensa viene dada por la distancia x recorrida.

Una política $\pi: S \rightarrow A$ es un mapeo de estados a acciones. El objetivo del aprendizaje por refuerzo es que el agente aprenda una política óptima basada en sus experiencias.

Una política es óptima si maximiza la recompensa a largo plazo, es decir, en muchos pasos.

Pero, ¿qué significa exactamente "maximizar la recompensa"? Definimos el valor, o la recompensa descontada

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (10.1)$$

de una política π cuando comenzamos en el estado inicial s_t . Aquí $0 \leq \gamma < 1$ es una constante, lo que asegura que la retroalimentación futura se descuenta más cuanto más lejos en el futuro ocurra. La recompensa inmediata r_t es la más ponderada. Esta función de recompensa es la más utilizada. Una alternativa que a veces resulta interesante es la recompensa media

$$V^\pi(s_t) = \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^{h-1} r_{t+i} \quad (10.2)$$

Una política π se llama óptima, si para todos los estados s

$$V^\pi(s) \geq V^{\pi'}(s). \quad (10.3)$$

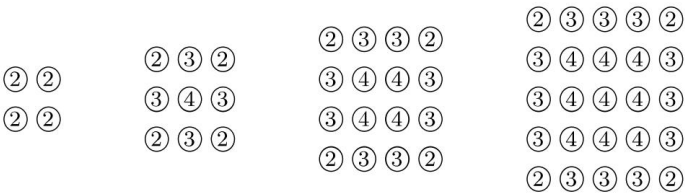


Fig. 10.5 El espacio de estado para el ejemplo con los valores 2, 3, 4, 5 para n_x y n_y . El número de acciones posibles se da para cada estado en los círculos respectivos

Tabla 10.2 Número de políticas para espacios de estado de diferentes tamaños en el ejemplo

n_x, n_y	Número de estados	Número de pólizas
2 2	24	$24 \times 24 = 576$
3 3	243	$243 \times 243 = 59049$
4 4	256	$256 \times 256 = 65536$
5 5	25	$25 \times 25 = 625$

Es decir, es al menos tan buena como todas las demás políticas de acuerdo con el valor definido función. Para una mejor legibilidad, la función de valor óptimo V^* se denotará V^* .

Los agentes discutidos aquí, o sus políticas, solo usan información sobre el estado actual para determinar el siguiente estado, y no el historial anterior. Esto se justifica si la recompensa de una acción solo depende del estado actual y las acciones actuales. Estos procesos se denominan procesos de decisión de Markov (MDP). En muchas aplicaciones, especialmente en robótica, no se conoce con exactitud el estado real del agente, lo que dificulta aún más la planificación de acciones. La razón de esto puede ser una señal de sensor ruidosa. A este proceso lo llamamos proceso de decisión de Markov parcialmente observable (POMDP).

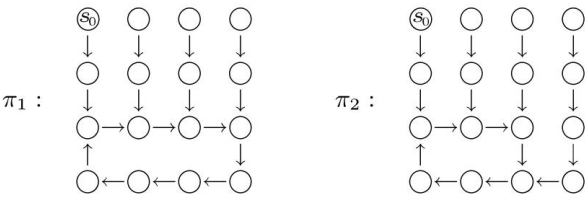
10.3 Búsqueda combinatoria no informada

La posibilidad más simple de encontrar una política exitosa es la enumeración combinatoria de todas las políticas, como se describe en el Cap. 6. Sin embargo, incluso en el ejemplo simple 10.1 hay muchas políticas, lo que hace que la búsqueda combinatoria se asocie con un costo computacional extremadamente alto. En la figura 10.5 se da el número de acciones posibles para cada estado. A partir de ahí, el número de pólizas posibles se calcula como el producto de los valores dados, como se muestra en la Tabla 10.2.

Para valores arbitrarios de n_x y n_y , siempre hay cuatro nodos de esquina con dos acciones posibles, $2(n_x - 2) + 2(n_y - 2)$ nodos de borde con tres acciones y $(n_x - 2)(n_y - 2)$ nodos internos con cuatro acciones. Así hay

$4 + 2 \cdot 2(n_x - 2) + 2(n_y - 2) + 4(n_x - 2)(n_y - 2)$

Fig. 10.6 Dos políticas diferentes para el ejemplo



Políticas diferentes para n_x y n_y fijos . El número de políticas crece exponencialmente con el número de estados. Esto es cierto en general si hay más de una acción posible por estado. Por lo tanto, para aplicaciones prácticas, este algoritmo es inútil. Incluso la búsqueda heurística, descrita en el Cap. 6, no se puede utilizar aquí. Dado que la recompensa directa de casi todas las acciones es cero, no se puede utilizar como una función de evaluación heurística.

El costo computacional se eleva aún más cuando consideramos que (además de s) debe enumerando todas las políticas), el valor V^π calcularse para cada s) generado debe cortarse política π y cada estado inicial s . La suma infinita en V^π para uso en un cálculo práctico; sin embargo, debido a la reducción exponencial de los factores γ en i (10.1) en la página 260, esto no presenta ningún problema.

En el Ejemplo 10.1 de la página 257 , la diferencia $x_{t+1} - x_t$ se puede usar como una recompensa inmediata por una acción en s_t , lo que significa que cada movimiento del cuerpo del robot hacia la derecha se recompensa con 1 y cada movimiento del cuerpo del robot hacia la izquierda está penalizada con -1 . En la figura 10.6 se muestran dos políticas. Aquí, la recompensa inmediata es cero en todas partes excepto en la fila inferior del espacio de estado. La política izquierda π_1 es mejor a largo plazo porque, para secuencias de acción largas, el progreso promedio por acción es $3/8 = 0,375$ para π_1 y $2/6 \approx 0,333$ para π_2 . Si usamos (10.1) s , el resultado es la siguiente tabla en la página 260 para V^π con el estado inicial s_0 en el arriba a la izquierda y varios valores de γ :

γ	0.9	0,8375 0,8
$V^{\pi_1}(s_0)$	2,52	1,156
$V^{\pi_2}(s_0)$	2,39	1,156

Aquí vemos que la política π_1 es superior a la política π_2 cuando $\gamma = 0,9$, y lo contrario es cierto cuando $\gamma = 0,8$. Para $\gamma \approx 0.8375$ ambas políticas son igualmente buenas. Podemos ver claramente que un mayor γ resulta en un mayor horizonte de tiempo para la evaluación de políticas.

10.4 Iteración de valor y programación dinámica

En el enfoque ingenuo de enumerar todas las políticas, se realiza mucho trabajo redundante, porque muchas políticas son en su mayor parte idénticas. Solo pueden diferir ligeramente. Sin embargo, cada política es completamente nueva generada y evaluada. Esto sugiere guardar resultados intermedios para partes de políticas y reutilizarlos.

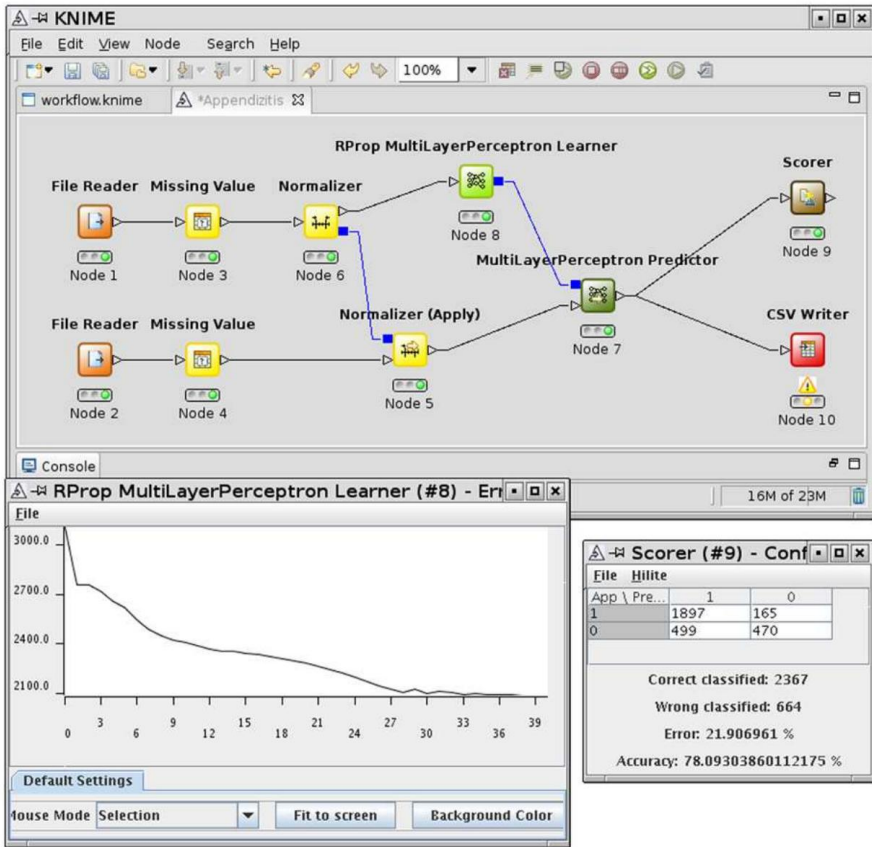


Fig. 8.36 La interfaz de usuario de KNIME con la ventana de flujo de trabajo, la curva de aprendizaje y la matriz de confusión

Una vez que el flujo está completamente construido y todos los nodos configurados, entonces un nodo arbitrario se puede ejecutar. Asegura automáticamente que los nodos predecesores se ejecuten, si necesario. El nodo J48 genera la vista del árbol de decisión, que se muestra a la derecha de la figura. Este árbol es idéntico al generado por C4.5 en la Secc. 8.4.5, aunque aquí el nodo $TRekt \leq 378$ se muestra colapsado.

A modo de comparación, un proyecto para aprender un perceptrón multicapa (ver Secc. 9.5) es se muestra en la figura 8.36. Esto funciona de manera similar a la percepción lineal presentada anteriormente, pero también puede dividir clases separables no linealmente. Aquí el flujo es algo mas complejo. Se necesita un nodo adicional para el manejo de errores de valores perdidos cuando preprocesamiento de cada uno de los dos archivos. Lo configuramos para que esas líneas se eliminen. Debido a que las redes neuronales no pueden manejar valores arbitrarios, los valores de todas las variables se escalan linealmente en el intervalo $[0, 1]$ utilizando el nodo "normalizador".

Después de aplicar el aprendizaje RProp, una mejora en la retropropagación (ver Secta. 9.5), podemos analizar la progresión en el tiempo del error de aproximación en la curva de aprendizaje mostrada. En la matriz de confusión, el anotador genera el análisis de los datos de prueba. El nodo "CSV Writer" en la parte inferior derecha sirve para exportar el archivos de resultados, que luego se pueden usar externamente para generar la curva ROC que se muestra en Fig. 7.10 en la página 142, para el cual lamentablemente no existe la herramienta KNIME (todavía).

En resumen, podemos decir lo siguiente sobre KNIME (y herramientas similares): para proyectos con requerimientos de análisis de datos no demasiado exóticos, vale la pena para trabajar con un banco de trabajo tan potente para el análisis de datos. El usuario ya guarda mucho tiempo con la etapa de preprocesamiento. Hay una gran selección de fácil uso "herramientas de minería", como clasificadores de vecinos más cercanos, algoritmos de aprendizaje de redes bayesianas simples, así como el algoritmo de agrupamiento k-means (Sección 8.7.2). Evaluación de los resultados, por ejemplo, la validación cruzada, puede llevarse a cabo fácilmente. queda por Cabe mencionar, que además de las mostradas, existen muchas otras herramientas para la visualización de los datos. Además, los desarrolladores de KNIME han hecho una extensión para Dispone de KNIME, con el que el usuario puede programar sus propias herramientas en Java o Python.

Sin embargo, también se debe mencionar que el usuario de este tipo de minería de datos El sistema debe traer consigo un conocimiento previo sólido del aprendizaje automático y el uso de técnicas de minería de datos. El software por sí solo no puede analizar los datos, pero en la mano de un especialista se convierte en una poderosa herramienta para la extracción de conocimiento a partir de datos. Para el principiante en el fascinante campo del aprendizaje automático, este sistema ofrece una oportunidad ideal y sencilla para poner a prueba los conocimientos de forma práctica y comparar los diversos algoritmos. El lector puede comprobar esto en el Ejercicio 8.21 de la página 220.

8.9 Resumen

Hemos cubierto a fondo varios algoritmos del campo establecido de aprendizaje supervisado, incluido el aprendizaje de árboles de decisión, las redes bayesianas y el más cercano. método vecino. Estos algoritmos son estables y se pueden usar de manera eficiente en varias aplicaciones y, por lo tanto, pertenecen al repertorio estándar en IA y minería de datos. El Lo mismo es cierto para los algoritmos de agrupamiento, que funcionan sin un "supervisor" y se pueden encontrar, por ejemplo, en aplicaciones de motores de búsqueda. Aprendizaje reforzado ya que otro campo del aprendizaje automático tampoco utiliza ningún supervisor. En contraste con el aprendizaje supervisado, donde el alumno recibe las acciones o respuestas correctas como etiquetas en los datos de entrenamiento, en el aprendizaje por refuerzo solo de vez en cuando se recibe retroalimentación positiva o negativa del entorno. En el cap. 10 mostraremos cómo esto obras. No es tan difícil la tarea en el aprendizaje semisupervisado, una subárea joven de aprendizaje automático, donde solo muy pocos de un gran número de datos de entrenamiento son etiquetado.

El aprendizaje supervisado es ahora un área bien establecida con muchas aplicaciones exitosas. Para el aprendizaje supervisado de datos con etiquetas continuas se puede emplear cualquier algoritmo de aproximación de función. Por lo tanto, hay muchos algoritmos de varios áreas de las matemáticas y la informática. Insecto. 9 presentaremos varios tipos de redes neuronales, algoritmos de mínimos cuadrados y máquinas de vectores de soporte,

que son todos aproximadores de funciones. Hoy en día, los procesos gaussianos son muy populares porque son muy universales y fáciles de aplicar y proporcionan al usuario una estimación de la incertidumbre de los valores de salida [RW06].

La siguiente taxonomía ofrece una descripción general de los algoritmos de aprendizaje más importantes y su clasificación.

Aprendizaje supervisado •

Aprendizaje perezoso

- k método vecino más cercano (clasificación + aproximación)
- Regresión ponderada localmente (aproximación)
- Aprendizaje basado en casos (clasificación + aproximación) • Aprendizaje

ávido – Inducción de

- árboles de decisión (clasificación)
- Aprendizaje de redes bayesianas (clasificación + aproximación)
- Redes neuronales (clasificación + aproximación)
- Procesos gaussianos (clasificación + aproximación)
- Wavelets, splines, funciones de base radial, . . .

Aprendizaje no supervisado (agrupación) •

Algoritmo del vecino más cercano •

Algoritmo del vecino más lejano • k-means • Redes

neuronales Aprendizaje

por refuerzo • Iteración de valor

• Aprendizaje Q •

Aprendizaje TD •

Métodos de

gradiente de políticas • Redes neuronales Sin embargo,

lo que se ha dicho sobre el aprendizaje supervisado solo es cierto, cuando se trabaja con un conjunto fijo de atributos conocidos. Un campo interesante, aún abierto bajo investigación intensiva, es la selección automática de funciones. Insecto. 8.4, para el aprendizaje con árboles de decisión, presentamos un algoritmo para el cálculo de la ganancia de información de atributos que ordena los atributos según su relevancia y utiliza solo aquellos que mejoran la calidad de la clasificación. Con este tipo de método es posible seleccionar automáticamente los atributos relevantes de un conjunto base potencialmente grande.

Este conjunto base, sin embargo, debe seleccionarse manualmente.

Todavía abierta y tampoco claramente definida está la cuestión de cómo la máquina puede encontrar nuevos atributos. Imaginemos un robot que se supone que debe recoger manzanas. Para ello debe aprender a distinguir entre manzanas maduras e inmaduras y otros objetos.

Tradicionalmente, determinamos ciertos atributos, como el color y la forma de las regiones de píxeles, y luego entrenamos un algoritmo de aprendizaje utilizando imágenes clasificadas manualmente.

También es posible que, por ejemplo, una red neuronal pueda entrenarse directamente con todos los píxeles de la imagen como entrada, lo que, sin embargo, para alta resolución está relacionado con graves problemas de tiempo de cálculo. En este caso, serían deseables enfoques que hagan sugerencias automáticamente para características relevantes. Pero esto sigue siendo ciencia ficción.

La agrupación proporciona un enfoque para la selección de características. Antes de entrenar la máquina de reconocimiento de manzanas, dejamos que la agrupación se ejecute en los datos. Para el aprendizaje (supervisado) de las clases apple y non apple, la entrada ya no son todos los píxeles, sino solo las clases encontradas durante la agrupación, potencialmente junto con otros atributos.

La agrupación en clústeres, en cualquier caso, se puede utilizar para el "descubrimiento" automático y creativo de funciones. Sin embargo, no está claro si las características descubiertas son relevantes.

El siguiente problema es aún más difícil: suponga que la cámara de video utilizada para el reconocimiento de Apple solo transmite imágenes en blanco y negro. La tarea ya no se puede resolver bien. Sería bueno que la máquina fuera creativa por sí misma, por ejemplo, sugiriendo que la cámara debería ser reemplazada por una cámara a color.

Esto sería pedir demasiado hoy.

Además de trabajos especializados sobre todos los subcampos del aprendizaje automático, existen excelentes libros de texto [Mit97, Bis06, Alp04, DHS01, HTF09]. Para conocer los resultados de la investigación actual, consulte el Journal of Machine Learning Research (<http://jmlr.csail.mit.edu>), el Machine Learning Journal, así como las actas de la Conferencia internacional sobre aprendizaje automático (ICML).) es recomendado. Para todos los desarrolladores de algoritmos de aprendizaje, el Repositorio de aprendizaje automático [DNM98] de la Universidad de California en Irvine (UCI) es interesante, con su gran colección de datos de prueba y entrenamiento para algoritmos de aprendizaje y herramientas de minería de datos. MLOSS, que significa software de código abierto de aprendizaje automático, es un excelente directorio de enlaces a software disponible gratuitamente (www.mloss.org).

8.10 Ejercicios

8.10.1 Introducción

Ejercicio 8.1 (a)

Especifique la tarea de un agente que debe predecir el clima para el día siguiente dados los valores medidos de temperatura, presión atmosférica y humedad. El clima debe clasificarse en una de las tres clases: soleado, nublado y lluvioso.

(b) Describa la estructura de un archivo con los datos de entrenamiento para este agente.

Ejercicio 8.2 Muestre que la matriz de correlación es simétrica y que todos los elementos de la diagonal son iguales a 1.

8.10.2 El perceptrón

Ejercicio 8.3 Aplicar la regla de aprendizaje del perceptrón a los conjuntos

$$M^+ = \{(0, 1, 8), (2, 0, 6)\} \text{ y } M^- = \{(-1, 2, 1, 4), (0, 4, -1)\}$$

del Ejemplo 8.2 en la página 172 y proporcione el resultado de los valores del vector de peso.

Ejercicio 8.4 Dada la tabla de la derecha con los datos de entrenamiento:

(a) Usando un gráfico, demuestre que los datos son linealmente separables. (b) Determinar manualmente los pesos w_1 y w_2 , así como el umbral Θ de un perceptrón (con umbral) que clasifica correctamente los datos.	número	x1	x2	Clase
	1	610		
	2	730		
	3	820		
	4	900		
	5	841		
(c) Programe la regla de aprendizaje del perceptrón y aplique su programa a la tabla. Compare los pesos descubiertos con los calculados manualmente.	6	861		
	7	921		
	8	951		

Ejercicio 8.5

(a) Dé una interpretación visual de la inicialización heurística

$$w_0 = \frac{x_i - x_i}{M+} - \frac{x_i - x_i}{M-}$$

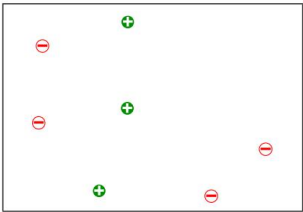
del vector de peso descrito en la Secc. 8.2.2.

(b) Dé un ejemplo de un conjunto de datos linealmente separable para el cual esta heurística no produzca una línea divisoria.

8.10.3 Método del vecino más cercano

Ejercicio 8.6

- (a) Muestre el diagrama de Voronoi para conjuntos de puntos vecinos.
- (b) Luego dibuje las líneas de división de clases.



Ejercicio 8.7 Sea dada la tabla con los datos de entrenamiento del ejercicio 8.4 . A continuación, utilice la distancia Manhattan $d(a, b)$, definida como $d(a, b) = |a_1 - b_1| + |a_2 - b_2|$, para determinar la distancia d entre dos puntos de datos $a = (a_1, a_2)$ y $b = (b_1, b_2)$. (a) Clasifique el vector $v = (8, 3.5)$ con el método del vecino más cercano. (b) Clasifique el vector $v = (8, 3.5)$ con el método del k vecino más cercano para $k = 2, 3, 5$.

Ejercicio 8.8

(a) Demuestre que en un espacio de características bidimensional es razonable, como se afirma en (8.3) en la página 180, ponderar los k vecinos más cercanos por el inverso de la distancia al cuadrado.

(b) ¿Por qué una ponderación usando $w_i = \frac{1}{1 + d(x, x_i)}$ tiene menos sentido?

8.10.4 Árboles de decisión

Ejercicio 8.9 Demuestre que la definición $0 \log_2 0 := 0$ es razonable, en otras palabras, que la función $f(x) = x \log_2 x$ se vuelve continua en el origen.

Ejercicio 8.10 Determine la entropía para las siguientes distribuciones.

- (a) $(1, 0, 0, 0, 0)$
- (b) $\frac{1}{2}, \frac{1}{2}, 0, 0, 0$
- (C) $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0, 0$
- (d) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}$
discreto discreto
- (mi) $\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}$
- (F) $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}, \frac{1}{32}, \dots$
discreto

Ejercicio 8.11

(a) Demuestre que las dos definiciones diferentes de entropía de (7.9) en la página 124 y la Definición 8.4 en la página 188 solo difieren en un factor constante, es decir, que

$$-\sum_{i=1}^n p_i \log_2 p_i = c$$

yo=1

$$-\sum_{i=1}^n p_i \ln p_i$$

yo=1

y dar la constante c. (b)

Demuestre que para el método MaxEnt y para los árboles de decisión no importa cuál de las dos fórmulas usamos.

Ejercicio 8.12 Desarrolle un árbol de decisión para el conjunto de datos D del Ejercicio 8.4 de la página

217. (a) Trate ambos atributos como discretos. (b) Ahora trate el atributo x2 como continuo y x1 como discreto. (c) Haga que C4.5 genere un árbol con ambas variantes. Use -m 1 -t 10 como parámetros para obtener diferentes sugerencias.

Ejercicio 8.13 Dado el siguiente árbol de decisión y tablas para datos de entrenamiento y prueba:

A

w

f

C

f

w

B

w

f

Datos de entrenamiento

Clase ABC

ttft

tfft

tftf

fftf

ffff

fttt

Datos de prueba

Clase ABC

ttft

tfft

ftff

fftf

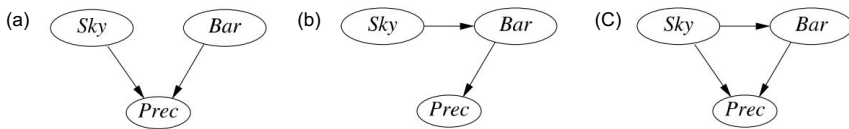
- (a) Dé la corrección del árbol para los datos de entrenamiento y prueba.
- (b) Dé una fórmula lógica proposicional equivalente al árbol. (c)
- Realice la poda en el árbol, dibuje el árbol resultante y dé su corrección.
- para los datos de entrenamiento y prueba.

Ejercicio 8.14

- (a) Al determinar el atributo actual, el algoritmo para generar árboles de decisión (Fig. 8.26 en la página 193) no elimina los atributos que ya se han utilizado más arriba en el árbol. A pesar de esto, un atributo discreto ocurre en una ruta como máximo una vez. ¿Por qué?
- (b) ¿Por qué los atributos continuos pueden ocurrir varias veces?

8.10.5 Aprendizaje de Redes Bayesianas

Ejercicio 8.15 Use la distribución dada en el Ejercicio 7.3 en la página 158 y determine los CPT para las tres redes bayesianas:



- (d) Determine la distribución para las dos redes de (a) y (b) y compare estos con la distribución original. ¿Qué red es "mejor"?
- (e) Ahora determine la distribución para la red (c). ¿Qué se te ocurre? justificar!

Ejercicio 8.16 Muestre que para las variables binarias $S_1, \dots, S_n$ y la variable de clase binaria K , una puntuación lineal de la forma

decisión = positivo si $w_1 S_1 + \dots + w_n S_n > \Theta$, negativo en
caso contrario

es igualmente expresivo en relación con el perceptrón y el clasificador ingenuo de Bayes, que deciden según la fórmula

decisión = positivo si $P(K|S_1, \dots, S_n) > 1/2$, negativo en
caso contrario

Ejercicio 8.17 En la implementación de la clasificación de texto con Naive Bayes, el subdesbordamiento del exponente puede ocurrir rápidamente porque los factores $P(w_i | K)$ (que aparecen en (8.10) en la página 206) suelen ser todos muy pequeños, lo que puede conducir a valores extremadamente pequeños. resultados. ¿Cómo podemos mitigar este problema?

Ejercicio 8.18 Escriba un programa para el análisis de texto bayesiano ingenuo. Luego entrénelo y pruébelo en puntos de referencia de texto usando una herramienta de su elección. Contar la frecuencia de las palabras en el texto se puede hacer fácilmente en Linux con el comando

```
gato <fechai> | tr -d "[:punto:]" | tr -s "[:espacio:]" "\n" | ordenar | uniq-ci
```

Obtenga los datos de los Veinte grupos de noticias de Tom Mitchell en la colección de pruebas comparativas de aprendizaje automático de UCI (Repositorio de aprendizaje automático) [DNM98]. Allí también encontrará una referencia a un programa Bayes ingenuo para la clasificación de texto de Mitchell.

8.10.6 Agrupación

Ejercicio 8.19 Muestre que para algoritmos que solo comparan distancias, aplicar una función f estrictamente monótonamente creciente a la distancia no hace ninguna diferencia. En otras palabras, debe demostrar que la distancia $d1(x, y)$ y la distancia $d2(x, y) := f(d1(x, y))$ conducen al mismo resultado con respecto a la relación de orden.

Ejercicio 8.20 Determinar las distancias d_s (producto escalar) de los siguientes textos entre sí. x_1 : Presentaremos la aplicación

de Naive Bayes al análisis de texto en un breve

texto de ejemplo de Alan Turing de [Tur50].

x_2 : Podemos esperar que las máquinas finalmente compitan con los hombres en todos los aspectos puramente campos intelectuales. Pero, ¿cuáles son los mejores para empezar?

x_3 : Nuevamente, no sé cuál es la respuesta correcta, pero creo que ambos enfoques deberían ser probado

8.10.7 Minería de datos

Ejercicio 8.21 Use KNIME (www.knime.de) y (a) Cargue el archivo

de ejemplo con los datos de Iris del directorio KNIME y experimente con las diversas representaciones de datos, especialmente con los diagramas de dispersión.

(b) Primero entrene un árbol de decisión para las tres clases y luego entrene una red RProp. (c) Cargue los datos de apendicitis en el sitio web de este libro. Compare la calidad de clasificación del método del k vecino más cercano con la de una red RProp. Optimice k así como el número de neuronas ocultas de la red RProp. (d) Obtenga un conjunto de datos de su elección de la recopilación de datos de

UCI para minería de datos en <http://kdd.ics.uci.edu> o para aprendizaje automático en <http://mlearn.ics.uci.edu/MLRepository.html> y experimentar con él.

Las redes neuronales son redes de células nerviosas en el cerebro de humanos y animales.

El cerebro humano tiene alrededor de 100 mil millones de células nerviosas. Nosotros, los humanos, debemos nuestra inteligencia y nuestra capacidad para aprender varias capacidades motoras e intelectuales a los complejos relés y la adaptabilidad del cerebro. Durante muchos siglos, biólogos, psicólogos y médicos han tratado de comprender cómo funciona el cerebro. Alrededor de 1900 se dio cuenta revolucionariamente de que estos diminutos componentes físicos del cerebro, las células nerviosas y sus conexiones, son responsables de la conciencia, las asociaciones, los pensamientos, la conciencia y la capacidad de aprender.

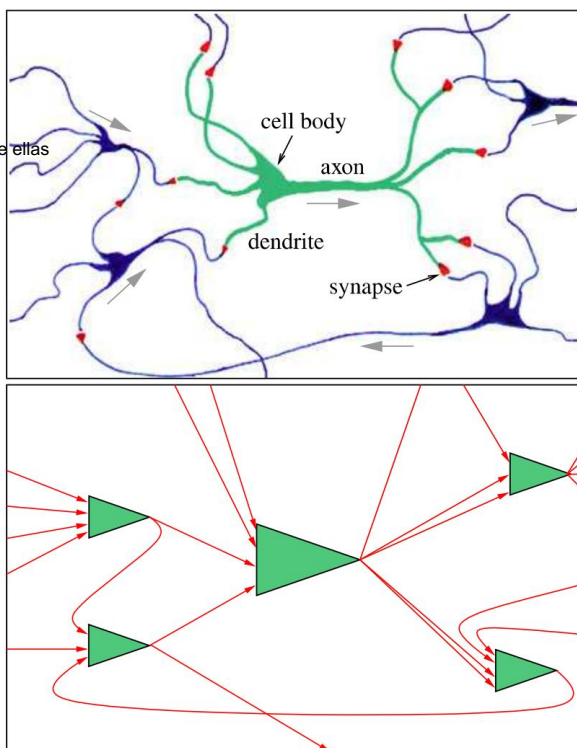
El primer gran paso hacia las redes neuronales en IA lo dieron McCulloch y Pitts en 1943 en un artículo titulado "Un cálculo lógico de las ideas inmanentes en la actividad nerviosa" [AR88]. Fueron los primeros en presentar un modelo matemático de la neurona como elemento básico de conmutación del cerebro. Este artículo sentó las bases para la construcción de redes neuronales artificiales y, por lo tanto, para esta rama tan importante de la IA.

Podríamos considerar el campo del modelado y simulación de redes neuronales como la rama de la biónica dentro de la IA.¹ Casi todas las áreas de la IA intentan recrear procesos cognitivos, como en la lógica o en el razonamiento probabilístico. Sin embargo, las herramientas utilizadas para el modelado, a saber, las matemáticas, los lenguajes de programación y las computadoras digitales, tienen muy poco en común con el cerebro humano. Con las redes neuronales artificiales, el enfoque es diferente. Partiendo del conocimiento sobre la función de las redes neuronales naturales, intentamos modelarlas, simularlas e incluso reconstruirlas en hardware. Todo investigador en esta área enfrenta el fascinante y emocionante desafío de comparar los resultados con el desempeño de los humanos.

En este capítulo intentaremos esbozar la progresión histórica definiendo un modelo de la neurona y su interconectividad, a partir de las ideas biológicas más importantes. Luego presentaremos varios modelos importantes y fundamentales: el modelo de Hopfield, dos modelos de memoria asociativa simples y el algoritmo de retropropagación, extremadamente importante en la práctica.

¹Bionics se preocupa por desbloquear los "descubrimientos de la naturaleza viva" y su conversión innovadora en tecnología [Wik10].

Fig. 9.1 Dos etapas del modelado de una red neuronal. Por encima de un modelo biológico y por debajo de un modelo formal con neuronas y conexiones dirigidas entre ellas.



9.1 De la biología a la simulación

Cada una de las aproximadamente 100 mil millones de neuronas en un cerebro humano tiene, como se muestra en una representación simplificada en la Fig. 9.1, la siguiente estructura y función. Además del cuerpo celular, la neurona tiene un axón, que puede establecer conexiones locales con otras neuronas sobre las dendritas. Sin embargo, el axón puede crecer hasta un metro de largo en forma de fibra nerviosa a través del cuerpo.

El cuerpo celular de la neurona puede almacenar pequeñas cargas eléctricas, de forma similar a un condensador o batería. Este almacenamiento se carga con los impulsos eléctricos entrantes de otras neuronas. Cuanto más impulso eléctrico entra, mayor es el voltaje. Si el voltaje supera un cierto umbral, la neurona se disparará. Esto significa que descarga su almacén, ya que envía un pico sobre el axón y las sinapsis. La corriente eléctrica se divide y llega a muchas otras neuronas sobre las sinapsis, en las que tiene lugar el mismo proceso.

Ahora surge la cuestión de la estructura de la red neuronal. Cada una de las aproximadamente 1011 neuronas en el cerebro está conectada a aproximadamente 1000 a 10 000 otras neuronas, lo que produce un total de más de 10^{14} conexiones. Si además consideramos que este número gigantesco de conexiones extremadamente delgadas está hecho de tejido blando tridimensional y que los experimentos en cerebros humanos no son fáciles de llevar a cabo, entonces queda claro por qué no tenemos un diagrama de circuito detallado de el cerebro. Presumiblemente

nunca seremos capaces de comprender completamente el diagrama de circuito de nuestro cerebro, basándonos únicamente en su inmenso tamaño.

Desde la perspectiva actual, ya no vale la pena intentar hacer un diagrama de circuito completo del cerebro, porque la estructura del cerebro es adaptativa. Cambia sobre la marcha y se adapta de acuerdo con las actividades del individuo y las influencias ambientales. El papel central aquí lo juegan las sinapsis, que crean la conexión entre las neuronas. En el punto de conexión entre dos neuronas, es como si dos cables se encontraran. Sin embargo, los dos conductores no están perfectamente conectados conductivamente, sino que hay un pequeño espacio que los electrones no pueden saltar directamente. Este vacío se llena con sustancias químicas, los llamados neurotransmisores. Estos pueden ser ionizados por un voltaje aplicado y luego transportar una carga sobre el espacio. La conductividad de este espacio depende de muchos parámetros, por ejemplo, la concentración y la composición química del neurotransmisor. Es esclarecedor que la función del cerebro reacciona de manera muy sensible a los cambios de esta conexión sináptica, por ejemplo, a través de la influencia del alcohol u otras drogas.

¿Cómo funciona el aprendizaje en una red neuronal de este tipo? Lo sorprendente aquí es que no son las unidades activas reales, a saber, las neuronas, las que son adaptativas, sino las conexiones entre ellas, es decir, las sinapsis. Específicamente, esto puede cambiar su conductividad. Sabemos que una sinapsis se fortalece por la cantidad de corriente eléctrica que debe transportar. Más fuerte aquí significa que la sinapsis tiene una conductividad más alta. Las sinapsis que se utilizan a menudo obtienen un peso cada vez mayor.

Para las sinapsis que se usan con poca frecuencia o que no están activas en absoluto, la conductividad continúa disminuyendo. Esto incluso puede llevarlos a morir.

Todas las neuronas del cerebro funcionan de forma asincrónica y en paralelo, pero, en comparación con una computadora, a muy baja velocidad. El tiempo de un impulso neural dura alrededor de un milisegundo, exactamente el mismo tiempo que tardan los iones en ser transportados a través del espacio sináptico. La frecuencia de reloj de la neurona está entonces por debajo de un kilohercio y, por lo tanto, es más baja que la de las computadoras modernas por un factor de 106. Esta desventaja, sin embargo, está más que compensada en muchas tareas cognitivas complejas, como el reconocimiento de imágenes, por la muy alta grado de procesamiento paralelo en la red de células nerviosas.

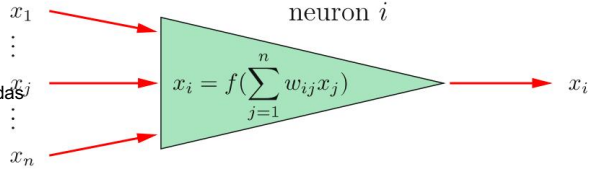
La conexión con el mundo exterior se produce a través de neuronas sensoras, por ejemplo en la retina de los ojos, o a través de células nerviosas con axones muy largos que van desde el cerebro hasta los músculos y pueden así realizar acciones como el movimiento de un pierna.

Sin embargo, aún no está claro cómo los principios discutidos hacen posible el comportamiento inteligente. Al igual que muchos investigadores en neurociencia, intentaremos explicar mediante simulaciones de un modelo matemático simple cómo se hacen posibles las tareas cognitivas, por ejemplo, el reconocimiento de patrones.

9.1.1 El modelo matemático

Primero reemplazamos el eje de tiempo continuo con una escala de tiempo discreta. La neurona i realiza el siguiente cálculo en un paso de tiempo. La "carga" de la activa-

Fig. 9.2 La estructura de una neurona formal, que aplica la función de activación f a la suma ponderada de todas las entradas



El potencial de conexión se logra simplemente mediante la suma de los valores de salida ponderados $x_1, \dots, x_n$ de todas las conexiones entrantes sobre la fórmula

$$\sum_{j=1}^n w_{ij} x_j$$

Esta suma ponderada es calculada por la mayoría de los modelos neuronales. Luego se le aplica una función de activación f y el resultado

$$x_i = f\left(\sum_{j=1}^n w_{ij} x_j\right)$$

se transmite a las neuronas vecinas como salida sobre los pesos sinápticos. En la Fig. 9.2 se muestra este tipo de neurona modelada. Para la función de activación hay varias posibilidades. La más simple es la identidad: $f(x) = x$. Por lo tanto, la neurona calcula solo la suma ponderada de los valores de entrada y la transmite. Sin embargo, esto conduce con frecuencia a problemas de convergencia con la dinámica neuronal porque la función $f(x) = x$ no tiene límites y los valores de la función pueden crecer más allá de todos los límites con el tiempo.

Muy bien restringida, por el contrario, es la función de umbral (función de paso de Heaviside)

$$HO(x) = \begin{cases} 0 & \text{si } x < \Theta, \\ 1 & \text{en caso contrario.} \end{cases}$$

Toda la neurona luego calcula su salida como

$$x_i = \begin{cases} 0 & \text{si } \sum_{j=1}^n w_{ij} x_j < \Theta, \\ 1 & \text{más} \end{cases}$$

Esta fórmula es idéntica a (8.1) en la página 173, es decir, a un perceptrón con el umbral Θ (Fig. 9.3 en la página 225). Las neuronas de entrada $1, \dots, n$ aquí sólo tienen la función de variables que transmiten sus valores establecidos externamente $x_1, \dots, x_n$ sin cambios.

La función de paso es bastante sensible para las neuronas binarias porque la activación de una neurona solo puede tomar los valores cero o uno de todos modos. En cambio, para continuo

Fig. 9.3 La neurona con función escalonada funciona como un perceptrón con un umbral

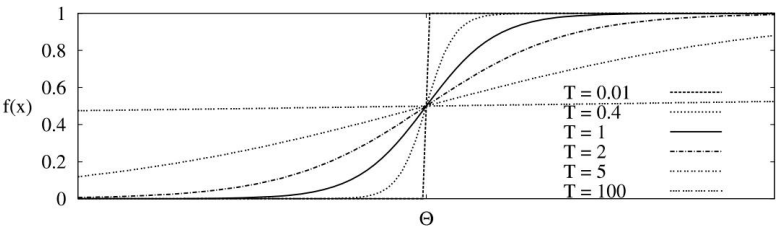
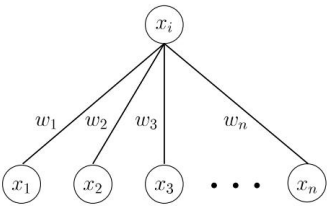


Fig. 9.4 La función sigmoidea para varios valores del parámetro $T \rightarrow 0$ resulta la función escalonada. Podemos ver que en el límite

neuronas con activaciones entre 0 y 1, la función de paso crea una discontinuidad. Sin embargo, esto se puede suavizar mediante una función sigmoidea, como

$$f(x) = \frac{1}{1 + e^{-\frac{x-\Theta}{T}}}$$

con el gráfico de la figura 9.4. Cerca del área crítica alrededor del umbral Θ , esta función se comporta casi linealmente y tiene un límite asintótico. El suavizado puede ser variado por el parámetro T .

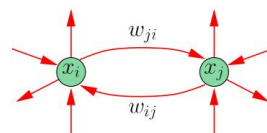
Modelar el aprendizaje es fundamental para la teoría de las redes neuronales. Como se mencionó anteriormente, una posibilidad de aprendizaje consiste en fortalecer una sinapsis de acuerdo con la cantidad de impulsos eléctricos que debe transmitir. Este principio fue postulado por D. Hebb en 1949 y se conoce como regla de Hebb:

Si hay una conexión w_{ij} entre la neurona j y la neurona i y se envían señales repetidas de la neurona j a la neurona i , lo que hace que ambas neuronas estén activas simultáneamente, entonces se refuerza el peso w_{ij} . Una posible fórmula para el cambio de peso w_{ij} es

$$w_{ij} = \eta x_i x_j$$

con la constante η (tasa de aprendizaje), que determina el tamaño de los pasos de aprendizaje individuales.

Fig. 9.5 Conexiones recurrentes entre dos neuronas en una red Hopfield



Hay muchas modificaciones de esta regla, que luego dan como resultado diferentes tipos de redes o algoritmos de aprendizaje. En las siguientes secciones, nos familiarizaremos con algunos de ellos.

9.2 Redes Hopfield

Mirando la regla de Hebb, vemos que para las neuronas con valores entre cero y uno, los pesos solo pueden crecer con el tiempo. No es posible que una neurona se debilite o incluso muera de acuerdo con esta regla. Esto se puede modelar, por ejemplo, mediante una constante de decaimiento que debilita un peso no utilizado en un factor constante por paso de tiempo, como 0,99.

Este problema se resuelve de manera bastante diferente con el modelo presentado por Hopfield en 1982 [Hop82]. Utiliza neuronas binarias, pero con los dos valores -1 para inactivo y 1 para activo. Usando la regla de Hebb obtenemos una contribución positiva al peso siempre que dos neuronas estén activas simultáneamente. Sin embargo, si solo una de las dos neuronas está activa, w_{ij} es negativa.

Las redes de Hopfield, que son un hermoso y visible ejemplo de memoria autoasociativa, se basan en esta idea. Los patrones se pueden almacenar en la memoria autoasociativa. Para recuperar un patrón guardado, basta con proporcionar un patrón similar. La tienda luego encuentra el patrón guardado más similar. Una aplicación clásica de esto es el reconocimiento de escritura a mano.

En la fase de aprendizaje de una red Hopfield, N patrones codificados en binario, guardados en los vectores $q^1, q^2, \dots, q^N$, se supone que deben ser aprendidos. Cada componente $q^k_j \in \{-1, 1\}$ vector q representa un píxel de un patrón. Para vectores que constan de n píxeles, se utiliza una red neuronal con n neuronas, una para cada posición de píxel. Las neuronas están completamente conectadas con la restricción de que la matriz de peso es simétrica y todos los elementos diagonales w_{ii} son cero. Es decir, no hay conexión entre una neurona y ella misma.

La red completamente conectada incluye bucles de retroalimentación complejos, los llamados recurrencias, en la red (Fig. 9.5).

Se pueden aprender N patrones simplemente calculando todos los pesos con la fórmula

$$w_{ij} = \frac{1}{N} \sum_{k=1}^N q^k_i q^k_j \quad (9.1)$$

Esta fórmula señala una relación interesante con la regla de Hebb. Cada patrón en el que los píxeles i y j tienen el mismo valor contribuye positivamente al peso w_{ij} .

Cada otro patrón hace una contribución negativa. Dado que cada píxel

corresponde a una neurona, aquí se refuerzan los pesos entre neuronas que simultáneamente tienen el mismo valor. Tenga en cuenta esta pequeña diferencia con la regla de Hebb.

Una vez que se han almacenado todos los patrones, la red se puede utilizar para el reconocimiento de patrones. Damos a la red un nuevo patrón x y actualizamos las activaciones de todas las neuronas en un proceso asíncrono según la regla

$$x_i = \begin{cases} -1 & \text{si } w_{ij} x_j < 0, j=1 \dots n \\ 1 & \text{demás} \end{cases} \quad (9.2)$$

hasta que la red se estabilice, es decir, hasta que no cambien más activaciones. Como esquema de programa, esto dice lo siguiente:

HOPFIELDASSOCIATOR(q)

Inicializar todas las neuronas: x

$= q$

Repetir $i = \text{Random}(1, n)$

Actualice la neurona i según (9.2)

Hasta que x converge

Retorno (x)

9.2.1 Aplicación a un ejemplo de reconocimiento de patrones

Aplicamos el algoritmo descrito a un ejemplo simple de reconocimiento de patrones. Debe reconocer dígitos en un campo de 10×10 píxeles. La red de Hopfield tiene entonces 100 neuronas con un total de

$$\frac{100 \cdot 99}{2} = 4950$$

pesos. Primero se entrenan los patrones de los dígitos 1, 2, 3, 4 en la Fig. 9.6 en la página 228 anterior. Es decir, los pesos se calculan mediante (9.1) en la página 226. Luego colocamos el patrón con ruido agregado y dejamos que la dinámica de Hopfield se desarrolle hasta la convergencia. En las filas 2 a 4 de la figura, se muestran cinco instantáneas del desarrollo de la red durante el reconocimiento. Con un 10 % de ruido, los cuatro patrones aprendidos se reconocen de forma muy fiable. Por encima de un 20% de ruido, el algoritmo converge con frecuencia a otros patrones aprendidos o incluso a patrones que no fueron aprendidos. Varios de estos patrones se muestran en la Fig. 9.6 en la página 228 a continuación.

Ahora guardamos los dígitos del 0 al 9 (Fig. 9.7 en la parte superior de la página 229) en la misma red y volvemos a probar la red con patrones que tienen una cantidad aleatoria de alrededor del 10 % de píxeles invertidos. En la figura, vemos claramente que la iteración de Hopfield a menudo no converge al estado aprendido más similar, incluso para solo un 10% de ruido. Evidentemente

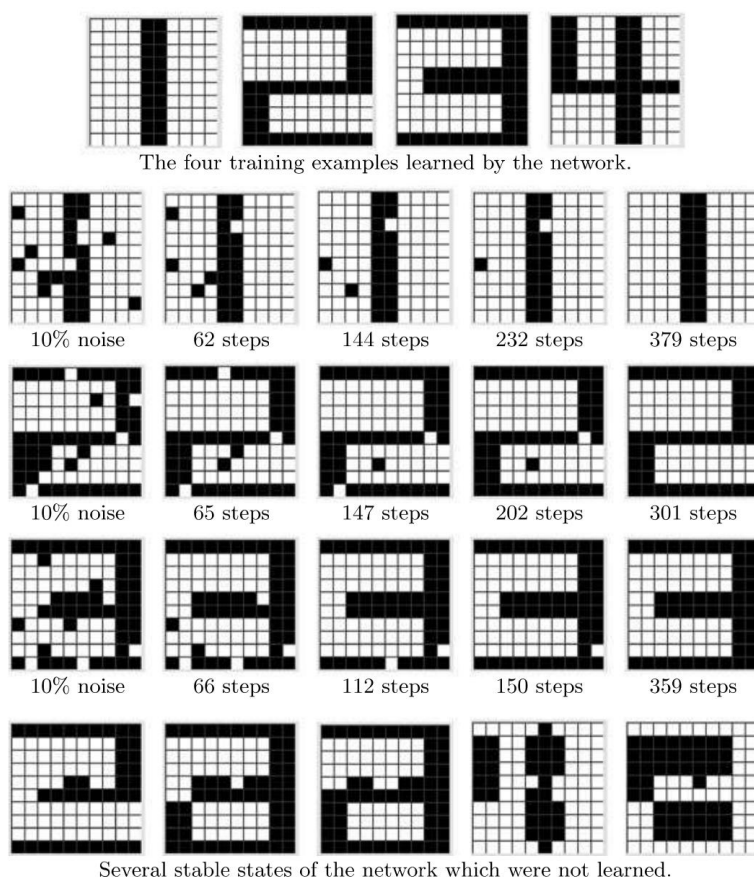


Fig. 9.6 Dinámica de una red de Hopfield. En las filas 2, 3 y 4 podemos ver fácilmente cómo converge la red y se reconoce el patrón aprendido después de unas 300 a 400 iteraciones. En la última fila se muestran varios estados estables que la red alcanza cuando el patrón de entrada se desvía demasiado de todos los patrones aprendidos.

la red puede guardar y reconocer de forma segura cuatro patrones, pero diez patrones superan su capacidad de memoria. Para entender esto mejor, echaremos un vistazo rápido a la teoría de esta red.

9.2.2 Análisis

En 1982, John Hopfield demostró en [Hop82] que este modelo es formalmente equivalente a un modelo físico de magnetismo. Pequeños imanes elementales, los llamados espines, se influyen mutuamente sobre sus campos magnéticos (ver Fig. 9.8 en la página 229). Si nosotros

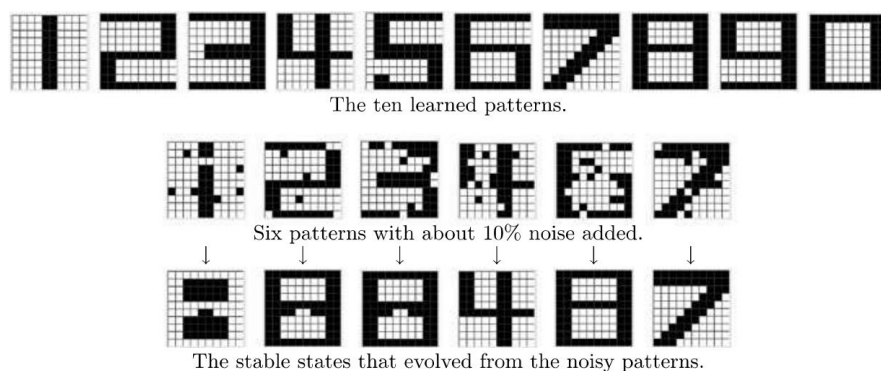


Fig. 9.7 Para diez estados aprendidos, la red muestra un comportamiento caótico. Incluso con poco ruido, la red converge a patrones incorrectos o a artefactos

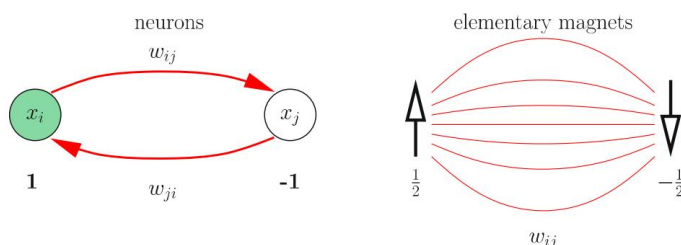


Fig. 9.8 Comparación entre la interpretación neural y física del modelo de Hopfield

observe dos de estos espines i y j , interactúan sobre una constante w_{ij} y la energía total del sistema es entonces

$$m_i = - \frac{1}{2} \sum_{j=y_0} w_{ij} x_j$$

Por cierto, $w_{ii} = 0$ también en física, porque las partículas no tienen autointeracción. Como las interacciones físicas son simétricas, $w_{ij} = w_{ji}$.

Un sistema físico en equilibrio adquiere un estado (estable) de energía mínima y, por lo tanto, minimiza $E(x, y)$. Si dicho sistema se lleva a un estado arbitrario, entonces se mueve hacia un estado de energía mínima. La dinámica de Hopfield definida en (9.2) en la página 227 corresponde exactamente a este principio porque actualiza el estado en cada iteración de manera que, de los dos estados -1 y 1 , se toma el de menor energía total. La contribución de la neurona i a la energía total es

$$- \frac{1}{2} \sum_{j=y_0} w_{ij} x_j$$

si ahora

$$\begin{aligned} & \text{---} \\ & w_{ij} x_j < 0, \\ & j = y_0 \end{aligned}$$

entonces $x_i = -1$ da como resultado una contribución negativa a la energía total y $x_i = 1$ da como resultado una contribución positiva. Para $x_i = -1$, la red toma un estado de menor energía que para $x_i = 1$. Análogamente, podemos afirmar que en el caso de

$$\begin{aligned} & \text{---} \\ & w_{ij} x_j \geq 0, \\ & j = y_0 \end{aligned}$$

debe ser cierto que $x_i = 1$.

Si cada iteración individual de la dinámica neuronal da como resultado una reducción de la función de energía, entonces la energía total del sistema disminuye monótonamente con el tiempo. Debido a que solo hay un número finito de estados, la red se mueve en el tiempo a un estado de energía mínima. Ahora tenemos la pregunta emocionante: ¿qué significan estos mínimos de la función de energía?

Como vimos en el experimento de reconocimiento de patrones, en el caso de pocos patrones aprendidos, el sistema converge a uno de los patrones aprendidos. Los patrones aprendidos representan mínimos de la función de energía en el espacio de estados. Sin embargo, si se aprenden demasiados patrones, entonces el sistema converge a mínimos que no corresponden a los patrones aprendidos. Aquí tenemos una transición de una dinámica ordenada a una caótica.

Hopfield y otros físicos han investigado exactamente este proceso y han demostrado que, de hecho, existe una transición de fase en un número crítico de patrones aprendidos. Si el número de patrones aprendidos excede este valor, entonces el sistema cambia de la fase ordenada a la caótica.

En la física magnética existe una transición del modo ferromagnético, en el que todos los imanes elementales intentan orientarse en paralelo, a un llamado vidrio giratorio, en el que los giros interactúan caóticamente. Un ejemplo más visible de una transición de fase física de este tipo es el derretimiento de un cristal de hielo. El cristal está en un alto estado de orden porque las moléculas de H₂O están estrictamente ordenadas. En el agua líquida, por el contrario, la estructura de las moléculas se disuelve y sus posiciones son más aleatorias.

En una red neuronal hay entonces una transición de fase desde el aprendizaje ordenado y el reconocimiento de patrones hasta el aprendizaje caótico en el caso de demasiados patrones, que ya no se pueden reconocer con certeza. Aquí definitivamente vemos paralelos con los efectos que ocasionalmente experimentamos nosotros mismos.

Podemos entender esta transición de fase [RMS92] si llevamos todas las neuronas a e insertamos estado del patrón, por ejemplo q^1 , los pesos aprendidos de (9.1) en la página 226, que es relevante en el término $\sum_j w_{ij} q_j^1$, para actualizar la neurona i . Esto da como resultado $j=1, j=i$ w_{ij} en

²Incluso el autor fue absorbido por esta ola, que lo llevó de la física a la IA en 1987.

Un problema especial de muchos modelos neuronales ya era evidente en el modelo de Hopfield. Incluso si existe una garantía de convergencia, no es seguro si la red convergerá a un estado aprendido o se atascará en un mínimo local. La máquina de Boltzmann, con valores de activación continuos y una regla de actualización probabilística para su dinámica de red, fue desarrollada como un intento de resolver este problema. Usando un parámetro de "temperatura", podemos variar la cantidad de cambios de estado aleatorios y así intentar escapar de los mínimos locales, con el objetivo de encontrar un mínimo global estable. Este algoritmo se denomina "recocido simulado". El recocido es un proceso de tratamiento térmico de metales con el objetivo de hacerlos más fuertes y más "estables".

El modelo de Hopfield realiza una búsqueda de un mínimo de la función de energía en el espacio de valores de activación. De este modo, encuentra el patrón guardado en los pesos, y que por lo tanto se representa en la función de energía. La dinámica de Hopfield también se puede aplicar a otras funciones de energía, siempre que la matriz de peso sea simétrica y los elementos diagonales sean cero. Esto fue demostrado con éxito por Hopfield y Tank en el problema del viajante de comercio [HT85, Zel94]. La tarea aquí es, dadas n ciudades y su matriz de distancia, encontrar el viaje de ida y vuelta más corto que visite cada ciudad exactamente una vez.

9.3 Memoria Asociativa Neural

Una memoria de lista tradicional puede ser, en el caso más simple, un archivo de texto en el que se guardan cadenas de dígitos línea por línea. Si el archivo está ordenado por línea, la búsqueda de un elemento se puede realizar muy rápidamente en tiempo logarítmico, incluso para archivos muy grandes.

Sin embargo, la memoria de lista también se puede utilizar para crear asignaciones. Por ejemplo, una guía telefónica es una asignación del conjunto de todos los nombres introducidos al conjunto de todos los números de teléfono. Esta asignación se implementa como una tabla simple, generalmente guardada en una base de datos.

El control de acceso a un edificio mediante reconocimiento facial es una tarea similar. Aquí también podríamos usar una base de datos en la que se guarde una foto de cada persona junto con el nombre de la persona y posiblemente otros datos. La cámara en la entrada luego toma una foto de la persona y busca en la base de datos una foto idéntica. Si se encuentra la foto, se identifica a la persona y se accede al edificio. Sin embargo, un edificio con tal sistema de control no recibiría muchos visitantes porque la probabilidad de que la foto actual coincida exactamente con la foto guardada es muy pequeña.

En este caso no basta con guardar la foto en una tabla. Más bien, lo que queremos es una memoria asociativa, que sea capaz no solo de asignar el nombre correcto a la foto, sino también a cualquiera de un conjunto potencialmente infinito de fotos "similares". Se debe generar una función para encontrar similitudes a partir de un conjunto finito de datos de entrenamiento, es decir, las fotos guardadas etiquetadas con los nombres. Un enfoque simple para esto es el método del vecino más cercano presentado en la Secc. 8.3. Durante el aprendizaje, todas las fotos simplemente se guardan.

Para aplicar esta función, se debe encontrar en la base de datos la foto más parecida a la actual. Para una base de datos con muchas fotos de alta resolución, este proceso, dependiendo de la métrica de distancia utilizada, puede requerir tiempos de cálculo muy largos y

por lo tanto, no se puede implementar de esta forma simple. Por lo tanto, en lugar de un algoritmo tan perezoso, preferiremos uno que transfiera los datos a una función que luego cree una asociación muy rápida cuando se aplique.

Encontrar una métrica de distancia adecuada presenta otro problema. Nos gustaría que una persona sea reconocida incluso si la cara de la persona aparece en otro lugar de la foto (traducción), o si es más pequeña, más grande o incluso girada. El ángulo de visión y la iluminación también pueden variar.

Aquí es donde las redes neuronales muestran sus puntos fuertes. Sin requerir que el desarrollador piense en una métrica de similitud adecuada, aún brindan buenos resultados. Presentaremos dos de los modelos de memoria asociativa más simples y comenzaremos con un modelo de Teuvo Kohonen, uno de los pioneros en esta área.

El modelo de Hopfield presentado en el capítulo anterior sería demasiado difícil de usar por dos razones. Primero, es solo una memoria autoasociativa, es decir, un mapeo aproximadamente idéntico que mapea objetos similares al original aprendido.

En segundo lugar, la compleja dinámica recurrente suele ser difícil de gestionar en la práctica.

Por lo tanto, ahora veremos redes simples de alimentación hacia adelante de dos capas.

9.3.1 Memoria de matriz de correlación

En [Koh72] Kohonen introdujo un modelo de memoria asociativa basado en álgebra lineal elemental. Esto asigna vectores de consulta $x \in \mathbb{R}^n$ a vectores de resultado $y \in \mathbb{R}^m$. Estamos buscando una matriz W que mapee correctamente un conjunto de datos de entrenamiento

$$T = \{(q_1, t_1), \dots, (q_n, t_n)\}$$

con N pares consulta-respuesta todos los vectores de consulta a sus respuestas.³ Es decir, para $p = 1, \dots, N$ debe darse el caso de que

$$t_p = W \cdot q_p, \quad (9.3)$$

o

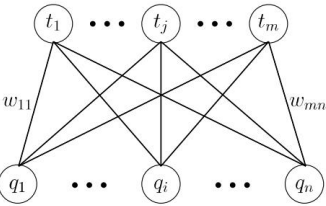
$$t_p = \sum_{j=1}^m w_{ij} \cdot q_{pj} \quad (9.4)$$

Para calcular los elementos de la matriz w_{ij} , la regla

$$w_{ij} = \sum_{p=1}^n q_{pj} t_{pi} \quad (9.5)$$

³Para una diferenciación clara entre los datos de entrenamiento y otros valores de una neurona, en la siguiente discusión nos referiremos al vector de consulta como q y a la respuesta deseada como t (objetivo).

Fig. 9.9 Representación de la memoria asociativa de Kohonen como una red neuronal de dos capas



se usa Estas dos ecuaciones lineales pueden entenderse simplemente como una red neuronal si definimos, como en la figura 9.9, una red de dos capas con q como capa de entrada y t como capa de salida. Las neuronas de la capa de salida tienen una función de activación lineal, y como regla de aprendizaje se utiliza (9.5) , que corresponde exactamente a la regla de Hebb.

Antes de mostrar que la red reconoce los datos de entrenamiento, necesitamos la siguiente definición:

Definición 9.1 Dos vectores x e y se llaman ortonormales si

$$T x \cdot y = \begin{cases} 1 & \text{si } x = y, \\ 0 & \text{en caso contrario.} \end{cases}$$

De este modo

Teorema 9.1 Si todos los N vectores de consulta q^p en los datos de entrenamiento son ortonormales, entonce cada vector q^p se asigna al vector objetivo t^p mediante la multiplicación con la matriz w_{ij} de (9.5) en la página 233.

Prueba Sustituimos (9.5) en la página 233 en (9.4) en la página 233 y obtenemos

$$\begin{aligned} (W \cdot q^{\text{pag}})_j &= \sum_{i=1}^n w_{ij} q_i^{\text{pag}} = \sum_{i=1}^n \sum_{r=1}^m w_{ij} t_r^{\text{pag}} q_i^{\text{pag}} \\ &= \sum_{j=1}^n q_j^{\text{pag}} \sum_{j=1}^n q_j^{\text{pag}} t_j^{\text{pag}} = \sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} t_j^{\text{pag}} \\ &= t_j^{\text{pag}} \sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} + \sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} t_j^{\text{pag}} \\ &= t_j^{\text{pag}} \underbrace{\left(\sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} \right)^T}_{=1} p + \sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} t_j^{\text{pag}} = t_j^{\text{pag}} \underbrace{\left(\sum_{j=1}^n q_j^{\text{pag}} q_j^{\text{pag}} \right)^T}_{=0} p + t_j^{\text{pag}} \end{aligned}$$

Por lo tanto, si los vectores de consulta son ortonormales, todos los patrones se asignarán correctamente a los objetivos respectivos. Sin embargo, la ortonormalidad es una restricción demasiado fuerte. Insecto. 9.4 presentaremos un enfoque que supera esta limitación.

Dado que las asignaciones lineales son continuas e inyectivas, sabemos que la asignación de vectores de consulta a vectores objetivo conserva la similitud. Por lo tanto, las consultas similares se asignan a objetivos similares debido a la continuidad. Sin embargo, al mismo tiempo sabemos que diferentes consultas se asignan a diferentes objetivos. Si la red fue entrenada para mapear caras a nombres y si el nombre Henry se asigna a una cara, entonces estamos seguros de que para la entrada de una cara similar, se producirá una salida similar a "Henry", pero "Henry" en sí mismo. se garantiza que no se calculará. Si la salida se puede interpretar como una cadena, entonces, por ejemplo, podría ser "Genry" o "Gfnry". Para llegar al caso aprendido más similar, Kohonen utiliza una codificación binaria para la neurona de salida. El resultado calculado de una consulta se redondea si su valor no es cero ni uno. Incluso entonces, no tenemos garantía de que alcanzaremos el vector objetivo. Alternativamente, podríamos agregar un mapeo posterior de la respuesta calculada al vector objetivo aprendido con la distancia más pequeña.

9.3.2 La pseudoinversa

Hay un enfoque diferente que podemos tomar para calcular la matriz de ponderación W : 1 pensando en todos los vectores de consulta como columnas de una matriz $n \times N$ $Q = (q_1, \dots, q_N)$ y $N, \dots, t$ de manera análoga, los vectores objetivo como columnas de una matriz $m \times N$ $T = (t_1, \dots, t_N)$. (t) Por lo tanto (9.3) en la página 233 se puede llevar a la forma

$$T = W \cdot Q. \quad (9.6)$$

Ahora intentamos resolver esta ecuación para W . Formalmente invertimos y obtenemos

$$W = T \cdot Q^{-1}. \quad (9.7)$$

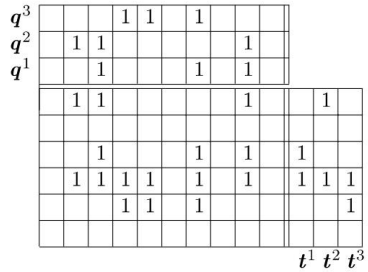
El requisito para esta conversión es la invertibilidad de Q . Para ello, la matriz debe ser cuadrada y debe constar de vectores columna linealmente independientes. Es decir, debe darse el caso de que $n = N$ y los n vectores de consulta deben ser linealmente independientes. Esta condición es de hecho inconveniente, pero aún no es tan estricta como la ortonormalidad requerida anteriormente. El aprendizaje por la regla de Hebb tiene la ventaja, sin embargo, de que podemos simplemente aplicarla incluso si los vectores de consulta no son ortonormales, con la esperanza de que el asociador funcione aceptablemente. Aquí, sin embargo, esto no es tan simple, ¿cómo podemos invertir una matriz no invertible?

Una matriz Q es invertible si y solo si existe una matriz Q^{-1} con la propiedad

$$Q \cdot Q^{-1} = I, \quad (9.8)$$

donde I es la matriz identidad. Si Q no es invertible (por ejemplo, porque Q no es cuadrado), entonces no existe una matriz Q^{-1} con esta propiedad. Sin embargo, ciertamente hay una matriz que se acerca a tener esta propiedad. En este sentido definimos

Fig. 9.10 La matriz W después de guardar tres pares (q^1, t^1) , (q^2, t^2) , (q^3, t^3) . Los campos vacíos corresponden al valor 0



Definición 9.2 Sea Q una matriz real $n \times m$. Una matriz $m \times n$ Q^+ se llama pseudoinversa a Q si minimiza

$$\|Q \cdot Q^+ - I\|$$

Aquí $\| \cdot \|$ es la norma euclidiana.

Ahora, usando

$$W = T \cdot Q^+ \tag{9.9}$$

podemos calcular una matriz de peso que minimice la diafonía (errores de asociación) $T - W \cdot Q$. Hay varias formas de calcular la pseudoinversa, por ejemplo, el método de los mínimos cuadrados que presentaremos en la Secc. 9.4.1.

9.3.3 La regla hebb binaria

En el contexto de la memoria asociativa, se sugirió la llamada regla binaria de Hebb. Requiere que el patrón esté codificado en binario. Esto significa que para todos los patrones $q \in \{0, 1\}^m$ y $t \in \{0, 1\}^n$. Además, la suma de (9.5) en la página 233 es 1) reemplazamos por un OR lógico simple y obtenemos la regla binaria de Hebb

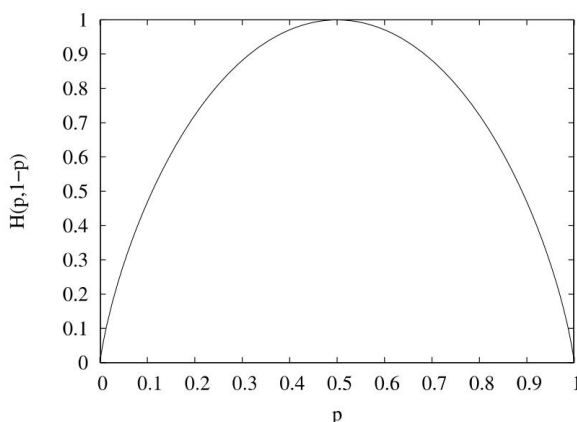
$$w_{ij} = p=1 \quad \text{pp} \quad \text{tq} \quad j_i \tag{9.10}$$

La matriz de pesos es también binaria, y un elemento de matriz w_{ij} es igual a uno si no es cero. y solo si al menos una de las entradas $q_j \in \{0, 1\}^m$, ..., $q_N \in \{0, 1\}^n$. Todas las demás matrices los elementos son cero Estamos tentados a creer que aquí se pierde mucha información durante el aprendizaje porque, cuando un elemento de la matriz toma el valor 1 una vez, no puede ser cambiado por patrones adicionales. La figura 9.10 muestra cómo se llena la matriz con unos para un ejemplo con $n = 10$, $m = 6$ después de aprender tres pares.

[illegible]

Es obvio que dicha memoria se "llena" con menos rapidez cuando los vectores de consulta y de destino están escasamente poblados con unos. Esta no es la única razón por la que la codificación de entrada y salida para memorias asociativas, como también para otras redes neuronales, es muy importante para un buen desempeño. Ahora demostraremos esto en una aplicación de esta memoria con codificaciones de entrada y salida adecuadamente elegidas.

Fig. 8.24 La función de entropía para el caso de dos clases. Vemos el máximo en $p = 1/2$ y la simetría con respecto al intercambio de p y $1 - p$



Definición 8.4 La Entropía H como métrica de la incertidumbre de una distribución de probabilidad se define por⁷

$$H(p) = H(p_1, \dots, p_n) := - \sum_{i=1}^n p_i \log_2 p_i.$$

Una derivación detallada de esta fórmula se encuentra en [SW76]. Si sustituimos el evento cierto $p = (1, 0, 0, \dots, 0)$, entonces $0 \log_2 0$, resulta una expresión indefinida. Resolvemos este problema por la definición $0 \log_2 0 := 0$ (vea el Ejercicio 8.9 en la página 218).

Ahora podemos calcular $H(1, 0, \dots, 0) = 0$. Mostraremos que la entropía en el hipercubo $[0, 1]^n$ bajo la restricción $\sum_{i=1}^n p_i = 1$ toma su valor máximo). En el caso de un evento con la distribución uniforme $\frac{1}{n}, \dots, \frac{1}{n}$ con dos posibles (resultados, que corresponden a dos clases, el resultado es

$$H(p) = H(p_1, p_2) = H(p_1, 1 - p_1) = -(p_1 \log_2 p_1 + (1 - p_1) \log_2 (1 - p_1)).$$

Esta expresión se muestra como una función de p_1 en la figura 8.24 con su máximo en $p_1 = 1/2$.

Debido a que a cada conjunto de datos clasificado D se le asigna una distribución de probabilidad p al estimar las probabilidades de clase, podemos extender el concepto de entropía a los datos por

⁷ En (7.9) en la página 124 se usa el logaritmo natural en lugar de $\log_2$ en la definición de entropía.

Debido a que aquí, y también en el caso del método MaxEnt, las entropías solo se comparan, esta diferencia no juega ningún papel. (ver Ejercicio 8.11 en la página 218).

la definición

$$H(D) = H(p).$$

Ahora, dado que el contenido de información $I(D)$ del conjunto de datos D pretende ser lo opuesto a la incertidumbre. Así definimos:

Definición 8.5 El contenido de información de un conjunto de datos se define como

$$I_o(D) := 1 - H(D). \quad (8.6)$$

8.4.3 Ganancia de información

Si aplicamos la fórmula de la entropía al ejemplo, el resultado es

$$H(6/11, 5/11) = 0,994$$

Durante la construcción de un árbol de decisión, el conjunto de datos se subdivide aún más por cada nuevo atributo. Cuanto más aumenta un atributo el contenido de información de la distribución al dividir los datos, mejor es ese atributo. Definimos en consecuencia:

Definición 8.6 La ganancia de información $G(D, A)$ a través del uso del atributo A está determinada por la diferencia del contenido de información promedio del conjunto de datos $D = D_1 \cup D_2 \cup \dots \cup D_n$ dividido por el atributo de valor n A y el contenido de información $I(D)$ del conjunto de datos no dividido, que produce

$$G(D, A) = \sum_{i=1}^n \frac{|D_i|}{|D|} I_o(D_i) - I_o(D).$$

Con (8.6) obtenemos de esto

$$\begin{aligned} G(D, A) &= \sum_{i=1}^n \frac{|D_i|}{|D|} I_o(D_i) - I_o(D) = \sum_{i=1}^n \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D)) \\ &= 1 - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i). \end{aligned} \quad (8.7)$$

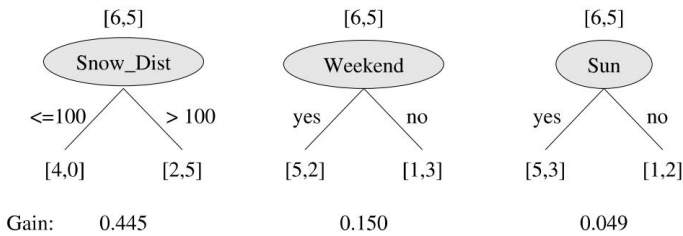


Fig. 8.25 La ganancia calculada para los diversos atributos refleja si la división de los datos por el atributo respectivo da como resultado una mejor división de clases. Cuanto más se desvían las distribuciones generadas por el atributo de la distribución uniforme, mayor es la ganancia de información

Aplicado a nuestro ejemplo para el atributo Snow_Dist, esto produce

$$\begin{aligned}
 G(D, \text{Snow\_Dist}) &= H(D) - \frac{4}{11} H_1(D \leq 100) - \frac{7}{11} H_1(D > 100) \\
 &= 0.994 - \frac{4}{11} \cdot 0 - \frac{7}{11} \cdot 0.863 = 0.445.
 \end{aligned}$$

Análogamente obtenemos

$$G(D, \text{fin de semana}) = 0.150$$

y

$$G(D, \text{Sol}) = 0.049.$$

El atributo Snow_Dist ahora se convierte en el nodo raíz del árbol de decisión. La situación de la selección de este atributo se aclara una vez más en la figura 8.25.

Los dos valores de atributo ≤ 100 y > 100 generan dos aristas en el árbol, que corresponden a los subconjuntos $D \leq 100$ y $D > 100$. Para el subconjunto $D \leq 100$ la clasificación es claramente sí. así el árbol termina aquí. En la otra rama $D > 100$ no hay un resultado claro. Por lo tanto, el algoritmo se repite recursivamente. De los dos atributos aún disponibles, Sol y Fin de semana, se debe elegir el mejor. Calculamos

$$G(D > 100, \text{fin de semana}) = 0.292$$

y

$$G(D > 100, \text{Sol}) = 0.170.$$

El nodo obtiene así el atributo Weekend asignado. Para Weekend = no, el árbol termina con la decisión Ski = no. Un cálculo de la ganancia aquí devuelve el valor 0. Para Weekend = yes, Sun da como resultado una ganancia de 0.171. Luego, la construcción del árbol finaliza porque no hay más atributos disponibles, aunque el ejemplo número 7 está clasificado erróneamente. El árbol terminado ya es familiar por la Fig. 8.23 en la página 186.

8.4.4 Aplicación de C4.5

El árbol de decisión que acabamos de generar también puede ser generado por C4.5. Los datos de entrenamiento se guardan en un archivo de datos ski.data en el siguiente formato:

```
<=100, si, si, si <=100, si, si, si
<=100, si, no, si <=100, no, si, si
>100, si, si, si >100, si, si, si >100,
si, si, no >100, si, no, no >100, no,
si, no >100, no, si, no >100, no,
```

no no

La información sobre atributos y clases se almacena en el archivo ski.names

(las líneas que comienzan con "|" son comentarios):

```
|Clases: no: no esquiar, si: ir a esquiar |
```

```
no si.
```

```
|
```

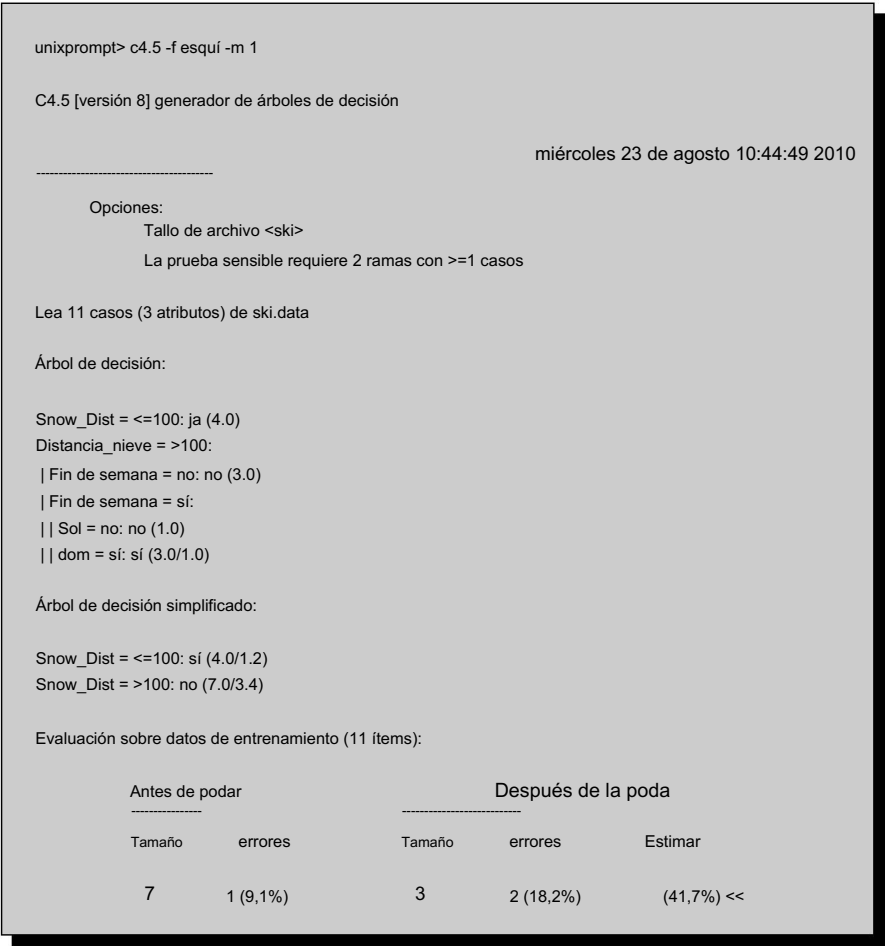
```
|Atributos |
```

```
Distancia_nieve:      <=100,>100.
```

```
Fin de semana:       no si.
```

```
Sol:                  no si.
```

A continuación, se llama a C4.5 desde la línea de comandos de Unix y se genera el árbol de decisión que se muestra a continuación, que se formatea con sangrías. La opción -f es para el nombre del archivo de entrada y la opción -m especifica la cantidad mínima de puntos de datos de entrenamiento necesarios para generar una nueva rama en el árbol. Debido a que el número de puntos de datos de entrenamiento en este ejemplo es extremadamente pequeño, -m 1 es sensato aquí. Para conjuntos de datos más grandes, se debe usar un valor de al menos -m 10.



Además, se proporciona un árbol simplificado con un solo atributo. Este árbol, que fue creado por la poda (ver Secc. 8.4.7), será importante para una cantidad creciente de datos de entrenamiento. En este pequeño ejemplo todavía no tiene mucho sentido. El error también se da la tasa para ambos árboles en los datos de entrenamiento. Los números entre paréntesis después de las decisiones, indique el tamaño del conjunto de datos subyacente y el número de errores. Por ejemplo, la línea Sun = yes: yes (3.0/1.0) en el árbol superior indica que para este nodo hoja Sun = yes, existen tres ejemplos de entrenamiento, uno de los cuales es clasificados falsamente. De este modo, el usuario puede leer si la decisión está fundamentada estadísticamente y/o es cierta.

En la Fig. 8.26 en la página 193 ahora podemos dar el esquema del algoritmo de aprendizaje para generar un árbol de decisión.

Ahora estamos familiarizados con los fundamentos de la generación automática de decisiones. árboles. Sin embargo, para la aplicación práctica, se necesitan ampliaciones importantes. Nosotros los presentará utilizando la ya conocida aplicación LEXMED .

ÁRBOL DE DECISIÓN GENERADO (Datos, Nodo)

A_{max} = Atributo con máxima ganancia de información Si

$G(A_{max}) = 0$

Entonces el Nodo se convierte en el nodo hoja con la clase más frecuente en

Datos De lo contrario asigne el atributo A_{max}

al Nodo Para cada valor $a_1, \dots, a_n$ de A_{max} , genere

un nodo sucesor: $K_1, \dots, K_n$

Divida los datos en $D_1, \dots, D_n$ con $D_i = \{x \mid \text{Data}[A_{max}(x)] = a_i\}$

Para todo $i \in \{1, \dots, n\}$

Si todo $x \in D_i$ pertenece a la misma clase C_i

Luego genera el nodo hoja K_i de la clase C_i

De lo contrario $\text{GENERATEDECISIONTREE}(D_i, K_i)$

Fig. 8.26 Algoritmo para la construcción de un árbol de decisión

8.4.5 Aprendizaje del diagnóstico de apendicitis

En el proyecto de investigación LEXMED, se desarrolló un sistema experto para el diagnóstico de apendicitis sobre una base de datos de pacientes [ES99, SE00]. El sistema, que trabaja con el método de máxima entropía, se describe en la Secc. 7.3

Ahora usamos la base de datos LEXMED para generar un árbol de decisiones para diagnosticar apendicitis con C4.5. Los síntomas utilizados como atributos se definen en el archivo app.names:

|Definición de las clases y atributos | Clases 0=apendicitis negativa |

1=apendicitis positivo 0,1. | Atributos | Edad: continua.

Sexo_(1=m___2=w): 1,2.

Dolor_Cuadrante1_(0=no___1=sí): 0,1.

Dolor_Cuadrante2_(0=no___1=sí): 0,1.

Dolor_Cuadrante3_(0=no___1=sí): 0,1.

Dolor_Cuadrante4_(0=no___1=sí): 0,1.

Vigilancia_local_(0=no___1=sí): 0,1.

Protección_generalizada_(0=no___1=sí): 0,1.

Rebote_ternura_(0=no___1=sí): 0,1.

Dolor_al_tapping_(0=no___1=sí): 0,1.

Dolor_durante_el_examen_rectal_(0=no__1=sí): 0,1.
Temp_axial: continuo.
Temporal: continuo.
Leucocitos: continuo.
Diabetes_mellitus_(0=no__1=sí): 0,1

Vemos que, además de muchos atributos binarios, como los diversos síntomas de dolor, también se presentan síntomas continuos, como la edad y la temperatura de la fiebre. En el siguiente archivo de datos de entrenamiento, app.data, en cada línea se describe un caso. En primera línea se encuentra un paciente masculino de 19 años con dolor en el tercer cuadrante (inferior derecho, donde está el apéndice), los dos valores de fiebre 36.2 y 37.8 grados centígrados, un valor de leucocitos de 13400 y un diagnóstico positivo, que es, un apéndice inflamado.

```
19,1,0,0,1,0,1,0,1,1,0,362,378,13400,0,1
13,1,0,0,1,0,1,0,1,1,1,383,385,18100 ,0,1
32,2,0,0,1,0,1,0,1,1,0,364,374,11800,0,1 18,2,0,0,1,1,0,0,0,0
,0,362,370,09300,0,0 73,2,1,0,1,1,1,0,1,1,1,376,380,13600,1,1
30,1,1,1,1,1,0,1 ,1,1,1,377,387,21100,0,1
56,1,1,1,1,1,0,1,1,1,0,390,?,14100,0,1 36,1,0,0,1
,0,1,0,1,1,0,372,382,11300,0,1
36,2,0,0,1,0,0,0,1,1,1,370,379,15300,0,1 33,1,0
0,1,0,1,0,1,1,0,367,376,17400,0,1
19,1,0,0,1,0,0,0,1,1,0,361,375,17600,0,1 12
,1,0,0,1,0,1,0,1,1,0,364,370,12900,0,0
...

```

Sin entrar en detalles sobre la base de datos, es importante mencionar que sólo se incluyen en la base de datos los pacientes que al llegar al hospital con sospecha de apendicitis fueron intervenidos quirúrgicamente. Vemos en la séptima fila que C4.5 también puede tratar con valores faltantes. Los datos contienen 9764 casos.

```
unixprompt> c4.5 -f aplicación -u -m 100

C4.5 [versión 8] generador de árboles de decisión                                     miércoles 23 de agosto 13:13:15 2006
-----

Lea 9764 casos (15 atributos) de app.data

Árbol de decisión:
```

```

Leucocitos <= 11030 :
| Rebote_ternura = 0:
|| Temp_rectal > 381 : 1 (135,9/54,2)
|| Temporal rectal <= 381 :
||| Protección_local = 0: 0 (1453.3/358.9)
||| Protección_local = 1:
||| | Sexo_(1=m___2=w) = 1: 1 (160.1/74.9)
||| | Sexo_(1=m___2=w) = 2: 0 (286.3/97.6)
| Rebote_ternura = 1:
|| Leucocitos <= 8600 :
|| | Temp_rectal > 378: 1 (176,0/59,4)
|| | Temporal rectal <= 378 :
|| | | Sexo_(1=m___2=w) = 1:
|| | | | Protección_local = 0: 0 (110,7/51,7)
|| | | | Protección_local = 1: 1 (160,6/68,5)
|| | | | Sexo_(1=m___2=w) = 2:
|| | | | Edad <= 14: 1 (131,1/63,1)
|| | | | Edad > 14 : 0 (398,3/137,6)
|| Leucocitos > 8600 :
|| | Sexo_(1=m___2=w) = 1: 1 (429.9/91.0)
|| | Sexo_(1=m___2=w) = 2:
|| | | Protección_local = 1: 1 (311.2/103.0)
|| | | Protección_local = 0:
|| | | | Temp_rectal <= 375: 1 (125,4/55,8)
|| | | | Temp_rectal > 375 : 0 (118,3/56,1)
Leucocitos > 11030 :
| Rebote_ternura = 1: 1 (4300.0/519.9)
| Rebote_ternura = 0:
|| Leucocitos > 14040 : 1 (826,6/163,8)
|| Leucocitos <= 14040 :
|| | Pain_on_tapping = 1: 1 (260.6/83.7)
|| | Dolor_al_tocar = 0:
|| | | Protección_local = 1: 1 (117,5/44,4)
|| | | Protección_local = 0:
|| | | | Temp_axial <= 368: 0 (131,9/57,4)
|| | | | Temp_axial > 368 : 1 (130,5/57,8)

```

Árbol de decisión simplificado:

```

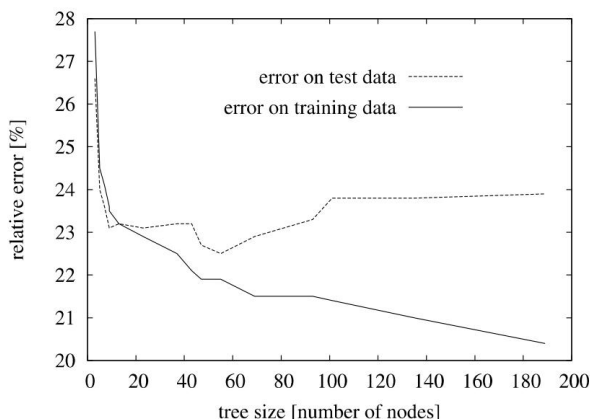
Leucocitos > 11030 : 1 (5767.0/964.1)
Leucocitos <= 11030 :
| Rebote_ternura = 0:
|| Temp_rectal > 381 : 1 (135,9/58,7)
|| Temporal rectal <= 381 :
||| Protección_local = 0: 0 (1453.3/370.9)
||| Protección_local = 1:
||| | Sexo_(1=m___2=w) = 1: 1 (160.1/79.7)
||| | Sexo_(1=m___2=w) = 2: 0 (286.3/103.7)
| Rebote_ternura = 1:
|| Leucocitos > 8600 : 1 (984,7/322,6)
|| Leucocitos <= 8600 :

```

268 2971 (b): clase 1

Para un atributo como el valor de leucocitos o la edad del paciente, una decisión basada en presumiblemente, una discretización binaria es demasiado imprecisa. Sin embargo, no hay necesidad de discretizar más finamente porque cada atributo continuo se prueba en cada recién generado nodo y, por lo tanto, puede ocurrir repetidamente en un árbol con un umbral diferente Θ, A . De este modo finalmente obtenemos una muy buena discretización cuya finura se ajusta al problema.

Fig. 8.27 Curva de aprendizaje de C4.5 sobre los datos de apendicitis. Vemos claramente el overfitting de árboles con más de 55 nodos



8.4.7 Poda: corte del árbol

Desde la época de Aristóteles se ha estipulado que de dos teorías científicas que explican igualmente bien una misma situación, se prefiere la más sencilla. Esta ley de la economía, también conocida ahora como la navaja de Occam, es de gran importancia para el aprendizaje automático y la minería de datos.

Un árbol de decisión es una teoría para describir los datos de entrenamiento. Una teoría diferente para describir los datos son los datos mismos. Si el árbol clasifica todos los datos sin ningún error, pero es mucho más compacto y, por lo tanto, más fácil de entender para los humanos, entonces es preferible según la navaja de Occam. Lo mismo es cierto para dos árboles de decisión de diferentes tamaños. Por lo tanto, el objetivo de cada algoritmo para generar un árbol de decisión debe ser generar el árbol de decisión más pequeño posible para una tasa de error dada. Entre todos los árboles con una tasa de error fija, siempre se debe seleccionar el árbol más pequeño.

Hasta ahora no hemos definido con precisión el término tasa de error. Como ya se mencionó varias veces, es importante que el árbol aprendido no solo memorice los datos de entrenamiento, sino que los generalice bien. Para probar la capacidad de generalización de un árbol, dividimos los datos disponibles en un conjunto de datos de entrenamiento y un conjunto de datos de prueba. Los datos de prueba están ocultos del algoritmo de aprendizaje y solo se usan para pruebas. Si se dispone de un gran conjunto de datos, como los datos de apendicitis, podemos, por ejemplo, utilizar dos tercios de los datos para el aprendizaje y el tercio restante para las pruebas.

Además de una mejor comprensibilidad, la navaja de Occam tiene otra justificación importante: la capacidad de generalización. Cuanto más complejo es el modelo (aquí un árbol de decisión), más detalles se representan, pero en la misma medida menos transferible es el modelo a nuevos datos. Esta relación se ilustra en la figura 8.27. Se entrenaron árboles de decisión de varios tamaños contra los datos de apendicitis. En el gráfico, se dan los errores de clasificación tanto en los datos de entrenamiento como en los datos de prueba. La tasa de error en los datos de entrenamiento disminuye monótonamente con el tamaño del árbol. Hasta un tamaño de árbol de 55 nodos, la tasa de error en los datos de prueba también disminuye. Sin embargo, si el árbol crece más, entonces la tasa de error comienza a aumentar nuevamente! Este efecto, que ya hemos visto en el método del vecino más próximo, se denomina sobreajuste.

Daremos este concepto, que es importante para casi todos los procesos de aprendizaje, una definición general tomada de [Mit97]:

Definición 8.7 Sea dado un algoritmo de aprendizaje específico, es decir, un agente de aprendizaje. Llamamos a un agente A sobreajustado a los datos de entrenamiento si hay otro agente A' cuyo error en los datos de entrenamiento es mayor que el de A , pero cuyo error en toda la distribución de datos es menor que el error de A .

¿Cómo podemos ahora encontrar este punto de error mínimo en los datos de prueba? El algoritmo más obvio se llama validación cruzada. Durante la construcción del árbol, el error en los datos de prueba se mide en paralelo. Tan pronto como el error aumenta significativamente, se guarda el árbol con el error mínimo. Este algoritmo es utilizado por el sistema CART mencionado anteriormente.

C4.5 funciona algo diferente. Primero, utilizando el algoritmo GENERATEDECISIONTREE de la Fig. 8.26 en la página 193, genera un árbol que generalmente está sobreajustado. Luego, mediante la poda, intenta eliminar los nodos del árbol hasta que el error en los datos de prueba, estimado por el error en los datos de entrenamiento, comienza a aumentar.⁸ Al igual que la construcción del árbol, este también es un algoritmo codicioso. . Esto significa que una vez que se poda un nodo, no se puede volver a insertar, incluso si esto luego resulta ser mejor.

8.4.8 Valores perdidos

Con frecuencia faltan valores de atributos individuales en los datos de entrenamiento. En el conjunto de datos LEX MED , se produce la siguiente entrada:

56, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14100, 0, 1,

en el que falta uno de los valores de fiebre. No obstante, dichos datos pueden utilizarse durante la construcción del árbol de decisión. Podemos asignar al atributo el valor más frecuente de todo el conjunto de datos o el más frecuente de todos los puntos de datos de la misma clase. Es incluso mejor sustituir la distribución de probabilidad de todos los valores de atributo por el valor de atributo faltante y dividir el ejemplo de entrenamiento en ramas de acuerdo con esta distribución. Por cierto, esta es una de las razones por las que aparecen valores no enteros en las expresiones entre paréntesis junto a los nodos hoja del árbol C4.5.

Los valores faltantes pueden ocurrir no solo durante el aprendizaje, sino también durante la clasificación. Estos se manejan de la misma manera que durante el aprendizaje.

⁸ Sería mejor usar el error en los datos de prueba directamente. Al menos cuando la cantidad de datos de entrenamiento sea suficiente para justificar un conjunto de pruebas separado.

8.4.9 Resumen

El aprendizaje de árboles de decisión es un enfoque favorito para las tareas de clasificación. Las razones de esto son su simple aplicación y velocidad. En un conjunto de datos de aproximadamente 10 000 puntos de datos LEXMED con 15 atributos cada uno, C4.5 requiere aproximadamente 0,3 segundos para aprender. Esto es muy rápido en comparación con otros algoritmos de aprendizaje.

Sin embargo, para el usuario también es importante que el árbol de decisión como modelo aprendido pueda ser entendido y potencialmente modificado. Tampoco es difícil transformar automáticamente un árbol de decisión en una serie de sentencias if-then-else y así construirlo eficientemente en un programa existente.

Debido a que se utiliza un algoritmo voraz para la construcción del árbol, así como durante la poda, los árboles son, en general, subóptimos. El árbol de decisión descubierto suele tener una tasa de error relativamente pequeña. Sin embargo, existe potencialmente un árbol mejor, porque la búsqueda codiciosa heurística de C4.5 prefiere árboles pequeños y atributos con alta ganancia de información en la parte superior del árbol. Para atributos con muchos valores, la fórmula presentada para la ganancia de información muestra debilidades. Las alternativas a esto se dan en [Mit97].

8.5 Aprendizaje de Redes Bayesianas

Insecto. 7.4, se mostró cómo construir una red bayesiana manualmente. Ahora introduciremos algoritmos para la inducción de redes bayesianas. Similar al proceso de aprendizaje descrito anteriormente, una red bayesiana se genera automáticamente a partir de un archivo que contiene datos de entrenamiento. Este proceso normalmente se descompone en dos partes.

1. Aprendizaje de la estructura de la red: para determinadas variables, la topología de la red se genera a partir de los datos de entrenamiento. Este primer paso es, con mucho, el más difícil y se le prestará más atención más adelante.
2. Aprendizaje de las probabilidades condicionales: para topologías de red conocidas, los CPT deben llenarse con valores. Si hay suficientes datos de entrenamiento disponibles, todas las probabilidades condicionales necesarias pueden estimarse contando las frecuencias en los datos.
Este paso se puede automatizar con relativa facilidad.
Ahora explicaremos cómo las redes bayesianas aprenden usando un algoritmo simple de [Jen01].

8.5.1 Aprendizaje de la estructura de la red

Durante el desarrollo de una red bayesiana (ver Secc. 7.4.6), se debe tener en cuenta la dependencia causal de las variables para obtener una red simple de buena calidad. El desarrollador humano confía en el conocimiento previo, que no está disponible para la máquina. Por lo tanto, este procedimiento no se puede automatizar fácilmente.

Encontrar una estructura óptima para una red bayesiana se puede formular como un problema de búsqueda clásico. Se da un conjunto de variables $V_1, \dots, V_n$ y un archivo con datos de entrenamiento. Estamos buscando un conjunto de aristas dirigidas sin ciclos entre los nodos.

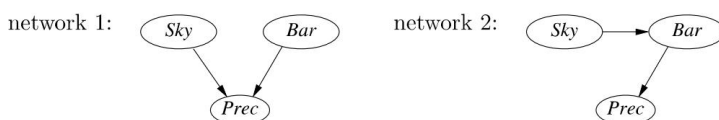


Fig. 8.28 Dos redes bayesianas para modelar el ejemplo de predicción meteorológica del ejercicio 7.3 de la página 158

$V_1, \dots, V_n$, es decir, un gráfico acíclico dirigido (DAG) que reproduce lo mejor posible los datos subyacentes.

Primero observamos el espacio de búsqueda. La cantidad de DAG diferentes crece más que exponencialmente con la cantidad de nodos. Para cinco nodos hay 29281 y para nueve nodos alrededor de 1015 DAG diferentes [MDBM00]. Por lo tanto, una búsqueda combinatoria no informada (ver Sección 6.2) en el espacio de todos los gráficos con un conjunto dado de variables es inútil si el número de variables crece. Por lo tanto, se deben utilizar algoritmos heurísticos. Esto plantea la cuestión de una función de evaluación para redes bayesianas. Es posible medir el error de clasificación de una red durante la aplicación a un conjunto de datos de prueba, como se hace, por ejemplo, en C4.5 (ver Secc. 8.4).

Para esto, sin embargo, las probabilidades calculadas por la red bayesiana deben mapearse en una decisión.

Se puede tomar una medida directa de la calidad de una red sobre la distribución de probabilidad. Suponemos que, antes de construir la red a partir de los datos, podríamos determinar (estimar) la distribución. Luego comenzamos la búsqueda en el espacio de todos los DAG, estimamos el valor de los CPT para cada DAG (es decir, para cada red bayesiana) usando los datos, y a partir de eso calculamos la distribución y la comparamos con la distribución conocida de los datos. Para la comparación de distribuciones obviamente necesitaremos una métrica de distancia.

Consideremos el ejemplo de predicción meteorológica del Ejercicio 7.3 de la página 158 con las tres variables Sky, Bar, Prec y la distribución

$$P(\text{Cielo}, \text{Barra}, \text{Prec}) = (0,40, 0,07, 0,08, 0,10, 0,09, 0,11, 0,03, 0,12).$$

En la figura 8.28 se presentan dos redes bayesianas, que ahora compararemos con respecto a su calidad. Cada una de estas redes hace un supuesto de independencia, que se valida cuando determinamos la distribución de la red y luego la comparamos con la distribución original (vea el ejercicio 8.15 en la página 219).

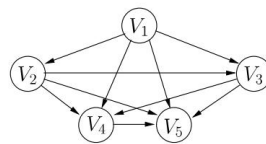
Como, para variables predeterminadas constantes, la distribución está claramente representada por un vector de longitud constante, podemos calcular la norma euclidiana de la diferencia de los dos vectores como una distancia entre distribuciones. Definimos

$$d_q(x, y) = \sum_i (x_i - y_i)^2$$

como la suma de los cuadrados de las distancias de los componentes del vector y calcule la distancia $d_q(P_a, P) = 0.0029$ de la distribución P_a de la red 1 de la distribución original. Para la red 2 calculamos $d_q(P_b, P) = 0.014$. Claramente, la red 1 es una mejor aproximación de la distribución. A menudo, en lugar de la distancia al cuadrado, la

Fig. 8.29 La red máxima con cinco variables

y aristas (V_i, V_j) que cumplen la condición $i < j$



la llamada distancia Kullback-Leibler

$$dk(x, y) = \sum_i y_i (\log_2 y_i - \log_2 x_i),$$

se utiliza una métrica de teoría de la información. Con él calculamos $dk(P_a, P) = 0,017$ y $dk(P_b, P) = 0,09$ y llegamos a la misma conclusión que antes. Es de esperar que las redes con muchos bordes se aproximen mejor a la distribución que aquellas con pocos bordes. Si se construyen todos los bordes de la red, se vuelve muy confuso y crea el riesgo de sobreajuste, como es el caso en muchos otros algoritmos de aprendizaje. Para evitar el sobreajuste, le damos a las redes pequeñas un mayor peso usando una función de evaluación heurística

$$f(N) = \text{Tamaño}(N) + w \cdot dk(P_N, P).$$

Aquí $\text{Size}(N)$ es el número de entradas en los CPT y P_N es la distribución de la red N . w es un factor de peso, que debe ajustarse manualmente.

El algoritmo de aprendizaje para redes bayesianas calcula la evaluación heurística $f(N)$ para muchas redes diferentes y luego elige la red con el valor más pequeño. Como se mencionó anteriormente, la dificultad consiste en la reducción del espacio de búsqueda para la topología de red que buscamos. Como un algoritmo simple es posible, a partir de una ordenación (por ejemplo causal) de las variables $V_1, \dots, V_n$, incluir en el gráfico solo aquellas aristas para las que $i < j$. Partimos del modelo maximal que cumple esta condición. Esta red se muestra en la figura 8.29 para cinco variables ordenadas.

Ahora, por ejemplo, en el espíritu de una búsqueda codiciosa (comparar Secc. 6.3.1), se elimina un borde tras otro hasta que el valor f ya no disminuye.

Este algoritmo no es práctico para redes más grandes en esta forma. El gran espacio de búsqueda, el ajuste manual de la ponderación w y la comparación necesaria con una distribución objetivo P son las razones de esto, ya que simplemente pueden volverse demasiado grandes o el conjunto de datos disponible puede ser demasiado pequeño.

De hecho, la investigación sobre el aprendizaje de redes bayesianas todavía está en pleno apogeo, y hay una gran cantidad de algoritmos sugeridos, por ejemplo, el algoritmo EM (ver Secc. 8.7.2), métodos de cadena de Markov Monte Carlo y muestreo de Gibbs [DHS01, Jor99, Jen01, HTF09]. Además del aprendizaje por lotes, que se ha presentado aquí, en el que la red se genera una vez a partir de todo el conjunto de datos, también existen algoritmos incrementales, en los que se utiliza cada nuevo caso individual para mejorar la red. También existen implementaciones de estos algoritmos, como Hugin (www.hugin.com) y Bayesware (www.bayesware.com).

8.6 El clasificador Naive Bayes

En la Fig. 7.14 en la página 152, el diagnóstico de apendicitis se modeló como una red bayesiana. Debido a que los bordes dirigidos comienzan en un nodo de diagnóstico y ninguno termina allí, se debe usar la fórmula de Bayes para responder una consulta de diagnóstico. Para los síntomas $S_1, \dots, S_n$ y el diagnóstico de valor k D con los valores $b_1, \dots, b_k$ calculamos la probabilidad

$$P(D|S_1, \dots, S_n) = P \frac{P(S_1, \dots, S_n|D) \cdot P(D)}{P(S_1, \dots, S_n)},$$

para el diagnóstico dados los síntomas del paciente. En el peor de los casos, es decir, si no hubiera variables independientes, sería necesario determinar todas las combinaciones de todos los síntomas y D para todas las 20 643 840 probabilidades de la distribución $P(S_1, \dots, S_n, D)$. Esto requeriría una enorme base de datos. En el caso de la red bayesiana de LEX MED, el número de valores necesarios (en los CPT) se reduce a 521. Sin embargo, la red se puede simplificar aún más, asumiendo que todas las variables de síntomas son condicionalmente independientes dado D , es decir:

$$P(S_1, \dots, S_n|D) = P(S_1|D) \cdots P(S_n|D).$$

La red bayesiana para la apendicitis luego se simplifica a la estrella que se muestra en la figura 8.30 en la página 203.

Así obtenemos la fórmula

$$P(S_1, \dots, S_n) = \frac{P(D) \prod_{i=1}^n P(S_i|D)}{P(S_1, \dots, S_n)}. \quad (8.8)$$

Las probabilidades calculadas se transforman en una decisión mediante un simple clasificador de Bayes ingenuo, que elige el máximo $P(D = d_i | S_1, \dots, S_n)$ de todos los valores d_i en D . Es decir, determina

$$d_{\text{Naive-Bayes}} = \operatorname{argmax}_i \{1, \dots, k\} P(D = d_i | S_1, \dots, S_n).$$

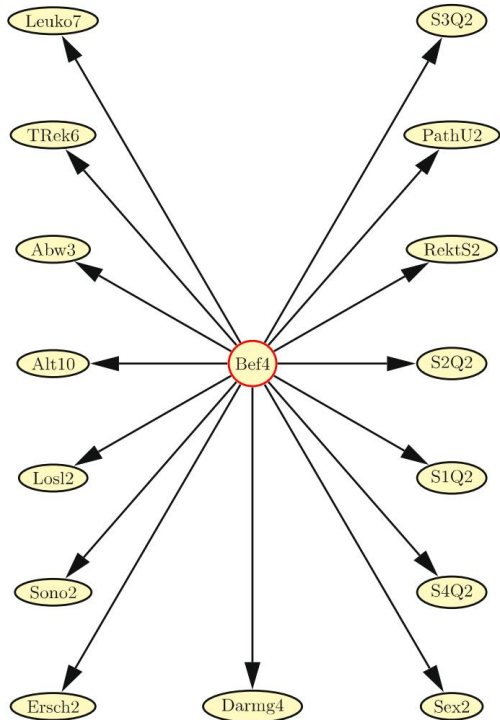
Debido a que el denominador en (8.8) es constante, se puede omitir durante la maximización, lo que da como resultado la ingenua fórmula de Bayes

$$d_{\text{Naive-Bayes}} = \operatorname{argmax}_i \{1, \dots, k\} P(D = d_i) \prod_{j=1}^n P(S_j | D).$$

Debido a que varios nodos ahora tienen menos ancestros, el número de valores necesarios para describir la distribución LEXMED en los CPT disminuye, según (7.22) en la página 151, a

$$6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 4 + 3 \cdot 4 + 10 \cdot (1 \cdot 4) + 1 = 141.$$

Fig. 8.30 Red bayesiana para la aplicación LEXMED con el supuesto de que todos los síntomas son condicionalmente independientes dado el diagnóstico



Para un sistema de diagnóstico médico como LEXMED esta simplificación no sería aceptable. Pero para tareas con muchas variables independientes, Naive Bayes es parcial o incluso muy adecuado, como veremos en el ejemplo de clasificación de texto.

Por cierto, la clasificación ingenua de Bayes es tan expresiva como el sistema de puntuación lineal descrito en la Secc. 7.3.1 (ver Ejercicio 8.16 en la página 219). Es decir, todas las puntuaciones comparten la suposición subyacente de que todos los síntomas son condicionalmente independientes dado el diagnóstico. Sin embargo, las partituras todavía se usan en medicina hoy en día. A pesar de que se generó a partir de una mejor base de datos representativa, la puntuación de Ohmann comparada con LEXMED en la Fig. 7.10 en la página 142 tiene una peor calidad de diagnóstico. Su limitada expresividad es sin duda una razón para ello. Por ejemplo, como con Naive Bayes, no es posible modelar dependencias entre síntomas usando

puntuaciones.

Estimación de probabilidades Si observamos la ingenua fórmula de Bayes en (8.8) en la página 202, vemos que toda la expresión se vuelve cero tan pronto como uno de los factores $P(S_i | D)$ en el lado derecho se vuelve cero. Teóricamente no hay nada malo aquí. En la práctica, sin embargo, esto puede conducir a efectos muy incómodos si los $P(S_i | D)$ son pequeños, porque estos se estiman contando frecuencias y sustituyéndolas

en

$$P(S_i = x | D = y) = \frac{|S_i = x \cap D = y|}{|D = y|}.$$

Suponga que para las variables S_i : $P(S_i = x | D = y) = 0.01$ y que hay 40 casos de entrenamiento con $D = y$. Entonces, con alta probabilidad, no hay caso de entrenamiento con $S_i = x$ y $D = y$, y estimamos $P(S_i = x | D = y) = 0$. Para un valor diferente de $D = z$, suponga que las relaciones están situadas de manera similar, pero la estimación da como resultado valores mayores que cero para todo $P(S_i = x | D = z)$. Por lo tanto, siempre se prefiere el valor $D = z$, que no refleja la distribución de probabilidad real. Por lo tanto, al estimar probabilidades, la fórmula

$$PAG(A|B) \approx \frac{|A \cap B|}{|B|} = \frac{\text{coger}}{\text{reemplazar datos}}$$

es reemplazado por

$$PAG(A|B) \approx \frac{n_{AB} + pf}{n_B + m},$$

donde $p = P(A)$ es la probabilidad a priori de A , y m es una constante de libre elección conocida como “tamaño de datos equivalente”. Cuanto mayor sea m , mayor será el peso de la probabilidad a priori en comparación con el valor determinado a partir de la frecuencia medida.

8.6.1 Clasificación de texto con Naive Bayes

Naive Bayes es muy exitoso y prolífico hoy en día en la clasificación de textos. Su aplicación principal, y al mismo tiempo muy importante, es el filtrado automático de correo electrónico en correos electrónicos deseados y no deseados o spam. En los filtros de spam como SpamAssassin [Sch04], entre otros métodos, se utiliza un clasificador Bayes ingenuo que aprende a separar los correos electrónicos deseados del spam. SpamAssassin es un sistema híbrido que realiza un filtrado inicial mediante listas blancas y negras. Las listas negras son listas de direcciones de correo electrónico bloqueadas de remitentes de spam cuyos correos electrónicos siempre se eliminan, y las listas blancas son aquellas con remitentes cuyos correos electrónicos siempre se entregan. Tras este prefiltrado, los correos electrónicos restantes son clasificados por el clasificador ingenuo de Bayes según su contenido real, es decir, según el texto. El valor de clase detectado luego se evalúa mediante una puntuación, junto con otros atributos del encabezado del correo electrónico, como el dominio del remitente, el tipo MIME, etc., y finalmente se filtra.

Aquí la capacidad de aprendizaje del ingenuo filtro Bayes es muy importante. Para ello el usuario debe primero clasificar manualmente una gran cantidad de correos electrónicos como deseados o spam. Luego se entrena el filtro. Para mantenerse actualizado, el filtro debe volver a entrenarse periódicamente. Para ello el usuario deberá clasificar correctamente todos los correos electrónicos que fueron clasificados erróneamente por el filtro, es decir, colocarlos en las carpetas correspondientes. Luego, el filtro se vuelve a entrenar continuamente con estos correos electrónicos.

Además del filtrado de spam, existen muchas otras aplicaciones para la clasificación automática de texto. Las aplicaciones importantes incluyen el filtrado de entradas no deseadas en foros de discusión de Internet y el seguimiento de sitios web con contenido delictivo, como actividades militantes o terroristas, pornografía infantil o racismo. También se puede utilizar para personalizar los motores de búsqueda para que se ajusten a las preferencias del usuario con el fin de clasificar mejor los resultados de la búsqueda. En el entorno industrial y científico, la búsqueda en toda la empresa en bases de datos o en la literatura está en primer plano de la investigación. A través de su capacidad de aprendizaje, un filtro puede adaptarse a los hábitos y deseos de cada usuario individual.

Presentaremos la aplicación de Naive Bayes al análisis de texto en un breve ejemplo. amplio texto de Alan Turing de [Tur50]:

"Podemos esperar que las máquinas eventualmente compitan con los hombres en todos los campos puramente intelectuales. Pero, ¿cuáles son los mejores para empezar? Incluso esta es una decisión difícil. Mucha gente piensa que una actividad muy abstracta, como jugar al ajedrez, sería lo mejor. También se puede sostener que es mejor dotar a la máquina de los mejores órganos sensoriales que el dinero pueda comprar, y luego enseñarle a entender y hablar inglés. Este proceso podría seguir la enseñanza normal de un niño. Las cosas se señalarían y nombrarían, etc. Nuevamente, no sé cuál es la respuesta correcta, pero creo que se deben probar ambos enfoques".

Supongamos que los textos como el que se da deben dividirse en dos clases: "I" para interesante y "¬I" para no interesante". Supongamos también que existe una base de datos de textos que ya están clasificados. ¿Qué atributos se deben usar? En un enfoque clásico para la construcción de una red bayesiana, definimos un conjunto de atributos como la longitud del texto, la longitud promedio de la oración, la frecuencia relativa de signos de puntuación específicos, la frecuencia de varias palabras importantes como "yo". , "máquinas", etc. Durante la clasificación utilizando Naive Bayes, por el contrario, se selecciona un algoritmo sorprendentemente primitivo. Para cada una de las n posiciones de palabra en el texto, se define un atributo s_i . Todas las palabras que aparecen en el texto se permiten como posibles valores para todas las posiciones s_i . Ahora para las clases I y ¬I los valores

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \prod_{s_i=1}^n P(s_i | I) \quad (8.9)$$

y $P(\neg I | s_1, \dots, s_n)$ y luego seleccionar la clase con el valor máximo. En el ejemplo anterior con un total de 113 palabras, esto produce

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \cdot P(s_1 = \text{"Nosotros"} | I) \cdot P(s_2 = \text{"podría"} | I) \cdots P(s_{113} = \text{"debería"} | I)$$

y

$$\begin{aligned} P(\neg I | s_1, \dots, s_n) &= c \cdot P(\neg I) \cdot P(s_1 = \text{"Nosotros"} | \neg I) \\ &\quad \cdot P(s_2 = \text{"puede"} | \neg I) \cdots P(s_{113} = \text{"debería"} | \neg I). \end{aligned}$$

El aprendizaje aquí es bastante simple. Las probabilidades condicionales $P(s_i | I)$, $P(s_i | \neg I)$ y las probabilidades a priori $P(I)$, $P(\neg I)$ simplemente deben calcularse. Ahora asumimos adicionalmente que los $P(s_i | I)$ no dependen de la posición en el texto. Este

significa, por ejemplo, que

$$P(s_{61} = "y" | I) = P(s_{69} = "y" | I) = P(s_{86} = "y" | I).$$

Podríamos entonces usar la expresión $P(y | I)$, con la nueva variable binaria y , como la probabilidad de que ocurra "y" en una posición arbitraria.

La implementación se puede acelerar un poco si encontramos la frecuencia n_i de cada palabra w_i que aparece en el texto y usa la fórmula

$$P(I | s_1, \dots, s_n) = c \cdot \prod_{y=1}^m P(w_i | y) n_i \quad (8.10)$$

lo que equivale a (8.9) en la página 205. Tenga en cuenta que el índice i en el producto solo corresponde al número I de diferentes palabras que aparecen en el texto.

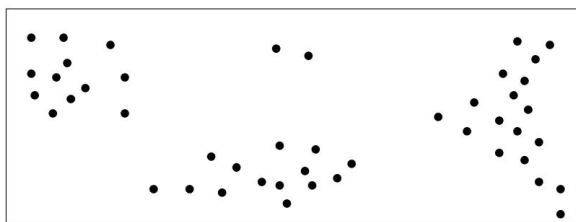
A pesar de su simplicidad, Naive Bayes ofrece excelentes resultados para la clasificación de textos. Los filtros de correo no deseado que funcionan con Naive Bayes logran tasas de error muy por debajo del uno por ciento. Los sistemas DSPAM y CRM114 pueden incluso estar tan bien entrenados que solo clasifican incorrectamente uno de cada 7000 u 8000 correos electrónicos respectivamente. Esto corresponde a una corrección de casi el 99,99 %.

8.7 Agrupación

Si buscamos en un buscador el término "marte", obtendremos resultados como "el planeta marte" y "conglomerado de chocolates, confitería y bebidas" que son semánticamente bastante diferentes. En el conjunto de documentos descubiertos hay dos grupos notablemente diferentes. Google, por ejemplo, todavía enumera los resultados de forma no estructurada. Sería mejor si el motor de búsqueda separara los grupos y se los presentara al usuario en consecuencia porque el usuario suele estar interesado en solo uno de los grupos.

La distinción del agrupamiento en contraste con el aprendizaje supervisado es que los datos de entrenamiento no están etiquetados. Por lo tanto, falta la preestructuración de los datos por parte del supervisor. Más bien, encontrar estructuras es el punto central de la agrupación. En el espacio de datos de entrenamiento, se encuentran acumulaciones de datos como las de la Fig. 8.31 en la página 207. En un conglomerado, la distancia de los puntos vecinos suele ser menor que la distancia entre puntos de diferentes conglomerados. Por tanto, la elección de una métrica de distancia adecuada para los puntos, es decir, para los objetos a agrupar y para los conglomerados, es de fundamental importancia. Como antes, asumimos a continuación que cada objeto de datos se describe mediante un vector de atributos numéricos.

Fig. 8.31 Ejemplo bidimensional simple con cuatro grupos claramente separados



8.7.1 Métricas de distancia

En consecuencia, para cada aplicación, las diversas métricas de distancia se definen para la distancia d entre dos vectores x e y en R^n . La más común es la distancia euclidiana.

$$d_e(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Algo más simple es la suma de distancias al cuadrado

$$d_q(x, y) = \left(\sum_{i=1}^n (x_i - y_i)^2 \right)^{1/2}$$

que, para algoritmos en los que solo se comparan distancias, es equivalente a la distancia euclidiana (Ejercicio 8.19 de la página 220). También se utilizan las ya mencionadas distancias de Manhattan.

$$d_m(x, y) = \sum_{i=1}^n |x_i - y_i|$$

así como la distancia del componente máximo

$$d_\infty(x, y) = \max_{i=1, \dots, n} |x_i - y_i|$$

que se basa en la norma máxima. Durante la clasificación del texto, la proyección normalizada de los dos vectores entre sí, es decir, el producto escalar normalizado

$$\frac{x \cdot y}{\|x\| \|y\|}$$

se calcula con frecuencia, donde $\|x\|$ es la norma euclidiana de x . Debido a que esta fórmula es una métrica para la similitud de los dos vectores, como métrica de distancia la inversa

$$d_s(x, y) = \frac{\|x\| \|y\|}{x \cdot y}$$

se puede utilizar, o ">" y "<" se pueden intercambiar para todas las comparaciones. En la búsqueda de un texto, los atributos $x_1, \dots, x_n$ se calculan de manera similar a Naive Bayes como componentes del vector x de la siguiente manera. Para un diccionario con 50.000 palabras, el valor x_i

es igual a la frecuencia de la i -ésima palabra del diccionario en el texto. Dado que normalmente casi todos los componentes son cero en dicho vector, durante el cálculo del producto escalar, casi todos los términos de la suma son cero. Al explotar este tipo de información, la implementación se puede acelerar significativamente (Ejercicio 8.20 en la página 220).

8.7.2 k-medias y el algoritmo EM

Siempre que el número de conglomerados ya se conozca de antemano, se puede utilizar el algoritmo k-means. Como su nombre lo indica, k clusters se definen por su valor promedio. Primero, los puntos medios del grupo k $\mu_1, \dots, \mu_k$ se inicializan de forma aleatoria o manual. Luego se llevan a cabo repetidamente los siguientes dos pasos: • Clasificación de todos los datos hasta su punto medio de conglomerado más cercano • Recálculo del punto medio del conglomerado.

El siguiente esquema resulta como un algoritmo:

```
K-MEANS( $x_1, \dots, x_n, k$ ) inicializa
 $\mu_1, \dots, \mu_k$  (p. ej. aleatoriamente)
Repetir
    clasificar  $x_1, \dots, x_n$  al  $\mu_i$  más cercano de cada
    uno recalcular  $\mu_1, \dots, \mu_k$ 
Hasta que no haya cambios en
 $\mu_1, \dots, \mu_k$  Devolver( $\mu_1, \dots, \mu_k$ )
```

El cálculo del punto medio del conglomerado μ para los puntos $x_1, \dots, x_l$ se realiza mediante

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i.$$

La ejecución de un ejemplo se muestra en la Fig. 8.32 en la página 209 para el caso de dos clases. Vemos cómo después de tres iteraciones, los centros de clase, que primero se eligieron al azar, se estabilizan. Si bien este algoritmo no garantiza la convergencia, por lo general converge muy rápidamente. Esto significa que el número de pasos de iteración suele ser mucho menor que el número de puntos de datos. Su complejidad es $O(ndkt)$, donde n es el número total de puntos, d la dimensionalidad del espacio de características y t el número de pasos de iteración.

En muchos casos, la necesidad de dar el número de clases por adelantado supone una inconveniente limitación. Por lo tanto, a continuación introduciremos un algoritmo que es más flexible.

Antes de eso, sin embargo, mencionaremos el algoritmo EM, que es una variante continua de k-means, ya que no hace una asignación firme de los datos a las clases, sino que para cada punto devuelve la probabilidad de que pertenezca a la varias clases.

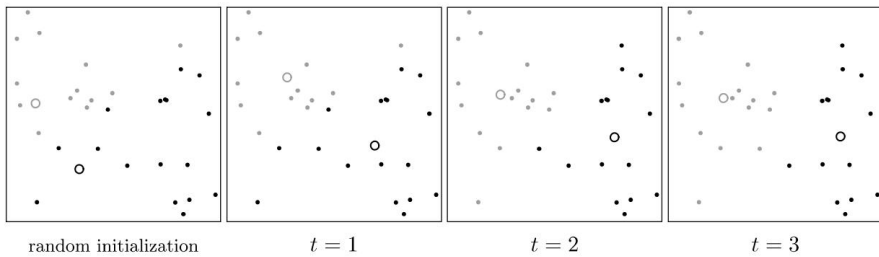


Figura 8.32 k-medias con dos clases ($k = 2$) aplicadas a 30 puntos de datos. El extremo izquierdo es el conjunto de datos con los centros iniciales, y a la derecha está el grupo después de cada iteración. Después de tres iteraciones se alcanza la convergencia

Aquí debemos suponer que se conoce el tipo de distribución de probabilidad. A menudo se utiliza la distribución normal. La tarea del algoritmo EM es determinar los parámetros (media μ_i y matrices de covarianza Σ_i de las k distribuciones normales multidimensionales) para cada grupo. De forma similar a k-means, se ejecutan repetidamente los dos pasos siguientes: Expectativa: Para cada punto de datos se calcula la probabilidad

$P(C_j | x_i)$ de que pertenezca a cada conglomerado.

Maximización: utilizando las probabilidades recién calculadas, se recalculan los parámetros de la distribución.

De este modo se consigue un agrupamiento más suave, que en muchos casos conduce a mejores resultados. Esta alternancia entre expectativa y maximización le da su nombre al algoritmo. Además de la agrupación, por ejemplo, el algoritmo EM se utiliza para aprender redes bayesianas [DHS01].

8.7.3 Agrupación jerárquica

En el agrupamiento jerárquico comenzamos con n grupos que consisten en un punto cada uno. Luego, los clústeres vecinos más cercanos se combinan hasta que todos los puntos se hayan combinado en un solo clúster, o hasta que se alcance un criterio de terminación. Obtenemos el esquema

```

CLUSTERING JERARQUICO( $x_1, \dots, x_n, k$ ) inicializar  $C_1$ 
=  $\{x_1\}, \dots, C_n = \{x_n\}$ 
Repetir
    Encontrar dos conglomerados  $C_i$  y  $C_j$  con la distancia más pequeña
    Combinar  $C_i$  y  $C_j$  Hasta
    que se alcance la condición de terminación
Regresar (árbol con conglomerados)
  
```

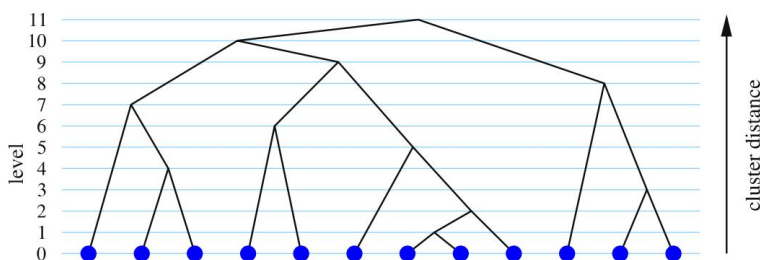


Fig. 8.33 En el agrupamiento jerárquico, los dos agrupamientos con la distancia más pequeña se combinan en cada paso

La condición de terminación podría elegirse como, por ejemplo, un número deseado de grupos o una distancia máxima entre grupos. En la figura 8.33, este algoritmo se representa esquemáticamente como un árbol binario, en el que de abajo hacia arriba en cada paso, es decir, en cada nivel, se conectan dos subárboles. En el nivel superior, todos los puntos se unifican en un gran grupo.

Hasta el momento no está claro cómo se calculan las distancias entre los grupos. De hecho, en la sección anterior definimos varias métricas de distancia para puntos, pero estas no se pueden usar en clústeres. Una métrica conveniente y de uso frecuente es la distancia entre los dos puntos más cercanos en los dos grupos C_i y C_j :

$$d_{\min}(C_i, C_j) = \min d(x, y) \text{ tal que } x \in C_i, y \in C_j$$

Así obtenemos el algoritmo del vecino más cercano, cuya aplicación se muestra en la Fig. 8.34 en la página 211.⁹ Vemos que este algoritmo genera un spanning mínimo. Además, el ejemplo muestra que los dos algoritmos descritos generan grupos bastante diferentes. Esto nos dice que para gráficos con clústeres que no están claramente separados, el resultado depende en gran medida del algoritmo o de la métrica de distancia elegida.

Para una implementación eficiente de este algoritmo, primero creamos una matriz de adyacencia en la que se guardan las distancias entre todos los puntos, lo que requiere $O(n^2)$ tiempo y memoria. Si el número de grupos no tiene un límite superior, el ciclo iterará $n - 1$ veces y el tiempo de cálculo asintótico se convierte en $O(n^3)$.

Para calcular la distancia entre dos cúmulos, también podemos usar la distancia entre los dos puntos más alejados

$$d_{\max}(C_i, C_j) = \max d(x, y) \text{ tal que } x \in C_i, y \in C_j$$

⁹El algoritmo del vecino más cercano no debe confundirse con el método del vecino más cercano para la clasificación de la Secc. 8.3.

¹⁰Un árbol de expansión mínima es un gráfico acíclico no dirigido con la suma mínima de longitudes de aristas.

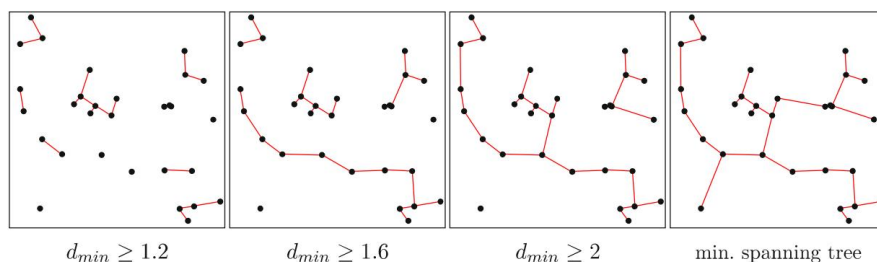


Fig. 8.34 El algoritmo del vecino más cercano aplicado a los datos de la Fig. 8.32 en la página 209 en diferentes niveles con 12, 6, 3, 1 conglomerados

y obtener el algoritmo del vecino más lejano. Alternativamente, se usa la distancia del punto medio del grupo $d\mu(C_i, C_j) = d(\mu_i, \mu_j)$. Además del algoritmo de agrupamiento presentado aquí, hay muchos otros, por lo que dirigimos al lector a [DHS01] para más estudio.

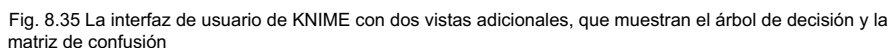
8.8 Minería de datos en la práctica

Todos los algoritmos de aprendizaje presentados hasta ahora se pueden utilizar como herramientas para la minería de datos. Para el usuario, sin embargo, a veces es bastante problemático acostumbrarse al nuevo software herramientas para cada aplicación y además poner los datos a analizar en el formato adecuado para cada caso particular.

Varios sistemas de minería de datos abordan estos problemas. La mayoría de estos sistemas ofrecen una cómoda interfaz gráfica de usuario con diversas herramientas para la visualización de los datos, para el preprocesamiento, como la manipulación de valores perdidos, y para el análisis. Para análisis, se utilizan los algoritmos de aprendizaje aquí presentados, entre otros.

La completa biblioteca Java de código abierto WEKA merece una mención especial. Ofrece una gran cantidad de algoritmos y simplifica el desarrollo de nuevos algoritmos.

El sistema de libre acceso KNIME, que presentaremos brevemente en el siguiente sección, ofrece una interfaz de usuario conveniente y todos los tipos de herramientas mencionado anteriormente. KNIME también utiliza módulos WEKA. Además ofrece un forma sencilla de controlar el flujo de datos de las herramientas de visualización, preprocesamiento y análisis elegidas con un editor gráfico. Mientras tanto, una gran cantidad de otros sistemas ofrecen una funcionalidad bastante similar, como el proyecto de código abierto [RapidMiner\(www.rapidminer.com\)](http://www.rapidminer.com), el sistema Clementine (www.spss.com/clementine) vendido por SPSS, y el marco analítico KXEN (www.kxen.com).



Usando los datos LEXMED , ahora mostraremos cómo extraer conocimiento de los datos usando KNIME (Konstanz Information Miner, www.knime.org). Primero generamos un árbol de decisión como se muestra en la figura 8.35. Después de crear un nuevo proyecto, se construye gráficamente un flujo de trabajo. Para hacer esto, las herramientas apropiadas simplemente se sacan del repositorio de nodos con el mouse y se arrastran a la ventana principal del flujo de trabajo.

Los datos de entrenamiento y prueba del archivo C4.5 se pueden leer con los dos nodos lectores de archivos sin ningún problema. Sin embargo, estos nodos también se pueden configurar fácilmente para otros formatos de archivo. El semáforo lateral debajo del nodo muestra su estado (no listo, configurado, ejecutado). Luego, se selecciona el nodo J48 de la biblioteca WEKA [WF01], que contiene una implementación Java de C4.5. La configuración para esto es bastante simple. Ahora se elige un nodo predictor, que aplica el árbol generado a los datos de prueba. Inserta una nueva columna en la tabla de datos de prueba "Predicción" con la clasificación generada por el árbol. A partir de ahí, el nodo marcador calcula la matriz de confusión que se muestra en la figura, que da el número de casos clasificados correctamente para ambas clases en la diagonal y, además, el número de puntos de datos falsos positivos y falsos negativos.

Tabla 8.1 Datos de entrenamiento para el clasificador de manzanas

Tamaño [cm]	8863. . .
Color	0,1 0,3 0,9 0,8 . . .
Clase de mercancía BAAB	. . .

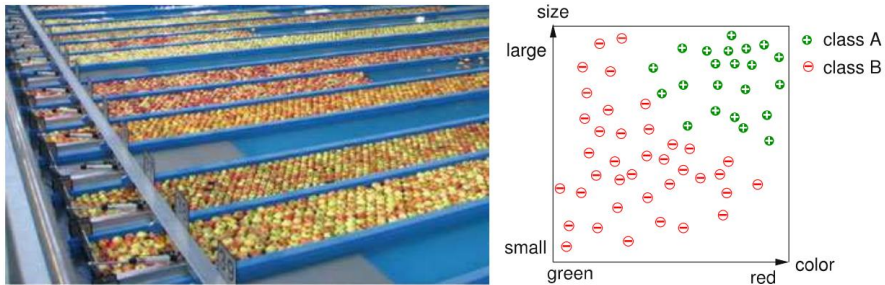
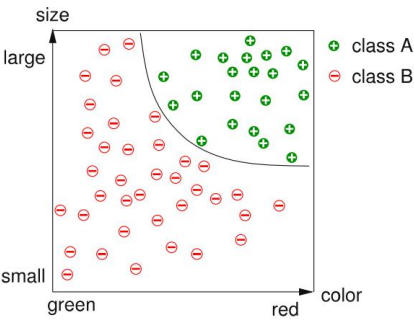


Fig. 8.2 Equipo de clasificación de manzanas de la empresa BayWa en Kressbronn y algunas manzanas clasificadas en las clases de mercancías A y B en el espacio de características (Foto: BayWa)

Fig. 8.3 La curva dibujada en el diagrama divide las clases y luego se puede aplicar a manzanas nuevas arbitrarias



Una visualización de los datos se muestra como puntos en un diagrama de dispersión a la derecha de la Fig. 8.2.

La tarea en el aprendizaje automático consiste en generar una función a partir de los datos recopilados y clasificados que calcula el valor de clase (A o B) para una nueva manzana a partir de las dos características tamaño y color. En la figura 8.3, dicha función se muestra mediante la línea divisoria trazada a lo largo del diagrama. Todas las manzanas con un vector de características en la parte inferior izquierda de la línea se colocan en la clase B y todas las demás en la clase A.

En este ejemplo, sigue siendo muy sencillo encontrar una línea divisoria de este tipo para las dos clases. Es claramente una tarea más difícil, y sobre todo mucho menos visualizable, cuando los objetos a clasificar se describen no solo por dos, sino por muchas características. En la práctica, se suelen utilizar 30 o más características. Para n características, la tarea consiste en encontrar un hiperplano $n - 1$ dimensional dentro del espacio de características n -dimensionales que divida las clases lo mejor posible. Una división “buena” significa que el porcentaje de objetos clasificados erróneamente es el menor posible.

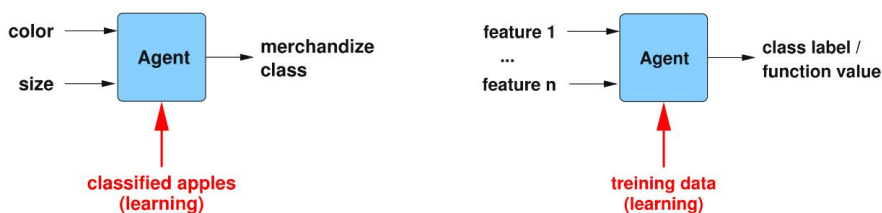


Fig. 8.4 Estructura funcional de un agente de aprendizaje para la clasificación de manzanas (izquierda) y en general (derecha)

Un clasificador asigna un vector de características a un valor de clase. Aquí tiene un número fijo, generalmente pequeño, de alternativas. La asignación deseada también se denomina función de destino. Si la función objetivo no corresponde a un dominio finito, entonces no se trata de una clasificación, sino de un problema de aproximación. Determinar el valor de mercado de una acción a partir de determinadas características es un problema de aproximación. En las siguientes secciones, presentaremos varios agentes de aprendizaje para ambos tipos de mapeos.

El agente de aprendizaje Podemos describir formalmente un agente de aprendizaje como una función que asigna un vector de características a un valor de clase discreto o, en general, a un número real. Esta función no está programada, sino que surge o cambia durante la fase de aprendizaje, influenciada por los datos de entrenamiento. En la Fig. 8.4, dicho agente se presenta en el ejemplo de clasificación de manzanas. Durante el aprendizaje, el agente se alimenta con los datos ya clasificados de la Tabla 8.1 en la página 163. A partir de entonces, el agente constituye un mapeo tan bueno como sea posible desde el vector de características hasta el valor de la función (por ejemplo, clase de mercancía).

Ahora podemos intentar acercarnos a una definición del término "aprendizaje automático".

Tom Mitchell [Mit97] da esta definición: Machine

Learning es el estudio de algoritmos informáticos que mejoran la automatización. a través de la experiencia.

Basándonos en esto, damos

Definición 8.1 Un agente es un agente de aprendizaje si mejora su desempeño (medido por un criterio adecuado) sobre datos nuevos y desconocidos a lo largo del tiempo (después de haber visto muchos ejemplos de entrenamiento).

Es importante probar la capacidad de generalización del algoritmo de aprendizaje sobre datos desconocidos, los datos de prueba. De lo contrario, todos los sistemas que acaban de guardar los datos de entrenamiento parecerían funcionar de manera óptima con solo recuperar los datos guardados. Un agente de aprendizaje se caracteriza por los siguientes términos: Tarea: la tarea del algoritmo de aprendizaje es aprender un mapeo. Esto podría ser, por ejemplo, el mapeo del tamaño y el color de una manzana a su clase de mercancía, pero también el mapeo de los 15 síntomas de un paciente a la decisión de extirpar o no su apéndice.



Figura 8.5 Minería de datos

Agente variable (más precisamente una clase de agentes): aquí tenemos que decidir con qué algoritmo de aprendizaje se va a trabajar. Si se ha elegido esto, se determina la clase de todas las funciones aprendibles.

Datos de entrenamiento (experiencia): los datos de entrenamiento (muestra) contienen el conocimiento que se supone que el algoritmo de aprendizaje debe extraer y aprender. Con la elección de los datos de entrenamiento, se debe asegurar que sea una muestra representativa de la tarea que se va a aprender.

Datos de prueba: importante para evaluar si el agente capacitado puede generalizar bien a partir de los datos de entrenamiento a nuevos datos.

Medida de rendimiento: para el dispositivo clasificador de manzanas, el número de manzanas clasificadas correctamente. Lo necesitamos para probar la calidad de un agente. Conocer la medida del desempeño suele ser mucho más fácil que conocer la función del agente. Por ejemplo, es fácil medir el rendimiento (tiempo) de un corredor de 10.000 metros. Sin embargo, esto no implica en absoluto que el árbitro que mide el tiempo pueda correr tan rápido. El árbitro solo sabe cómo medir el desempeño, pero no la "función" del agente cuyo desempeño está midiendo.

¿Qué es la minería de datos? La tarea de una máquina de aprendizaje para extraer conocimiento de los datos de entrenamiento. A menudo, el desarrollador o el usuario quiere que la máquina de aprendizaje haga que el conocimiento extraído también sea legible para los humanos. Todavía es mejor si el desarrollador puede incluso alterar el conocimiento. El proceso de inducción de árboles de decisión en la Secc. 8.4 es un ejemplo de este tipo de método.

Desafíos similares provienen del comercio electrónico y la gestión del conocimiento.

Aquí se presenta un problema clásico: a partir de las acciones de los visitantes de su portal web,

al dueño de un negocio en Internet le gustaría crear una relación entre el características de un cliente y la clase de productos que son interesantes para ese cliente. Luego, un vendedor podrá colocar anuncios específicos del cliente. Este se demuestra de manera elocuente en www.amazon.com, donde se recomienda al cliente productos similares a los vistos en la visita anterior. En muchos En las áreas de publicidad y marketing, así como en la gestión de las relaciones con los clientes (CRM), se están empezando a utilizar técnicas de minería de datos. Siempre que grandes cantidades de datos están disponibles, se puede intentar utilizar estos datos para el análisis de clientes preferencias para mostrar anuncios específicos del cliente. El campo emergente de aprendizaje de preferencia se dedica a este propósito.

El proceso de adquisición de conocimiento a partir de datos, así como su representación. y aplicación, se denomina minería de datos. Los métodos utilizados suelen tomarse a partir de estadísticas o aprendizaje automático y debe ser aplicable a muy grandes cantidades de datos a un costo razonable.

En el contexto de la adquisición de información, por ejemplo en Internet o en una intranet, la minería de textos juega un papel cada vez más importante. Las tareas típicas incluyen encontrar texto similar en un motor de búsqueda o la clasificación de textos, que por ejemplo se aplica en filtros de spam para correo electrónico. Insecto. 8.6.1 introduciremos la generalizada Algoritmo de Bayes ingenuo para la clasificación de texto. Un desafío relativamente nuevo para La minería de datos es la extracción de información estructural, estática y dinámica de estructuras gráficas como redes sociales, redes de tráfico o tráfico de Internet.

Debido a que las dos tareas descritas de aprendizaje automático y minería de datos son formalmente muy similares, los métodos básicos utilizados en ambas áreas son en su mayor parte idénticos. Por lo tanto, en la descripción de los algoritmos de aprendizaje, no se hará ninguna distinción entre el aprendizaje automático y la minería de datos.

Debido al enorme impacto comercial de las técnicas de minería de datos, ahora existen muchas optimizaciones sofisticadas y toda una línea de potentes sistemas de minería de datos, que ofrecen una gran paleta de herramientas convenientes para la extracción de conocimiento de datos. Tal sistema se introduce en la Secc. 8.8.

8.1 Análisis de datos

Las estadísticas proporcionan varias formas de describir los datos con parámetros simples. De estos elegimos algunos que son especialmente importantes para el análisis de la formación datos y probarlos en un subconjunto de los datos LEXMED de la Secc. 7.3. En este ejemplo conjunto de datos, los síntomas $x_1, \dots, x_{15}$ de $N = 473$ pacientes, descritos de manera concisa en la Tabla 8.2 en la página 167, así como la etiqueta de clase, es decir, el diagnóstico (apendicitis positivo/negativo)—se enumeran. El paciente número uno, por ejemplo, es descrito por el vector

$$^1x_- = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37, 9, 38, 8, 23100, 0, 1)$$

Tabla 8.2 Descripción de variables x1,...,x16.
un poco diferente
La formalización se utilizó en
Tabla 7.2 en la página 133

Var. núm.	Descripción	Valores
1	Edad	Continuo
2	Sexo (1=masculino, 2=femenino)	1, 2
3	Dolor cuadrante 1	0, 1
4	Dolor cuadrante 2	0, 1
5	Dolor cuadrante 3	0, 1
6	Dolor cuadrante 4	0, 1
7	Defensa muscular local	0, 1
8	Defensa muscular generalizada 0, 1	
9	Sensibilidad de rebote	0, 1
10	Dolor al tocar	0, 1
11	Dolor durante el examen rectal	0, 1
12	Temperatura axial	Continuo
13	Temperatura rectal	Continuo
14	leucocitos	Continuo
15	Diabetes mellitus	0, 1
desviada	Apendicitis	0, 1

y el paciente número dos por

$2x_{-} = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36,9, 37,4, 8100, 0, 0)$

El paciente número dos tiene el valor de leucocitos x $\frac{2}{14} = 8100$.

Para cada variable xi , su promedio $\bar{x}_i$ se define como

$$\bar{x}_i := \frac{1}{norte} \sum_{pag=1}^{norte} X_i^{(pag)}$$

y la desviación estándar si como una medida de su desviación promedio del promedio
valor como

$$s_i := \sqrt{\frac{1}{norte - 1} \sum_{pag=1}^{norte} (x_{i,pag} - \bar{x}_i)^2}$$

La cuestión de si dos variables xi y xj son estadísticamente dependientes (correlacionadas) es importante para el análisis de datos multidimensionales. por ejemplo, el covarianza

$$\sigma_{ij} = \frac{1}{norte - 1} \sum_{pag=1}^{norte} (x_{i,pag} - \bar{x}_i)(x_{j,pag} - \bar{x}_j)$$

Tabla 8.3 Matriz de correlación para las 16 variables de apendicitis medidas en 473 casos

1.	-0,009 0,14	0,037 -0,096 0,12	0,018 0,051 -0,034 -0,041 0,034 0,037 0,05 -0,037 0,37	0,012
-0,009 1.	-0,0074 -0,019 -0,06 0,063 -0,17	0,0084 -0,17 -0,14 -0,13 -0,017 -0,034 -0,14	0,045 -0,2	
0,14 -0,0074 1.	0,55 -0,091 0,24	0,13 0,24 0,045 0,18 0,028 0,02 0,045 0,03	0,11 0,045	
0,037 -0,019 0,55	1. -0,24 0,33	0,051 0,25 0,074 0,19 0,087 0,11	0,12 0,11 0,14 -0,0091	
-0,096 -0,06 -0,091 -0,24	1. 0,059 0,14	0,034 0,14 0,049 0,057 0,064 0,058 0,11	0,017 0,14	
0,12 0,063 0,24	0,33 0,059 1.	0,071 0,19 0,086 0,15 0,048 0,11	0,12 0,063 0,21 0,053	
0,018 -0,17	0,13 0,051 0,14 0,071 1.	0,16 0,4 0,28 0,2 0,24 0,36	0,29 -0,0001 0,33	
0,051 0,0084 0,24	0,25 0,034 0,19	0,16 1. 0,17 0,23 0,24 0,19 0,24 0,27	0,083 0,084	
-0,034 -0,17	0,045 0,074 0,14 0,086 0,4	0,17 1. 0,53 0,25 0,19 0,27 0,27	0,026 0,38	
-0,041 -0,14	0,18 0,19 0,049 0,15	0,28 0,23 0,53 1. 0,24 0,15 0,19 0,23	0,02 0,32	
0,034 -0,13	0,028 0,087 0,057 0,048 0,2	0,24 0,25 0,24 1. 0,17 0,17 0,22	0,098 0,17	
0,037 -0,017 0,02	0,11 0,064 0,11	0,24 0,19 0,19 0,15 0,17 1.	0,72 0,26 0,035 0,15	
0,05 -0,034 0,045	0,12 0,058 0,12	0,36 0,24 0,27 0,19 0,17 0,72 1.	0,38 0,044 0,21	
-0,037 -0,14	0,03 0,11 0,11 0,063 0,29	0,27 0,27 0,23 0,22 0,26 0,38 1.	0,051 0,44	
0,37 0,045	0,11 0,14 0,017 0,21 -0,0001 0,083	0,026 0,02 0,098 0,035 0,044 0,051	1. -0,0055	
0,012 -0,2	0,045 -0,0091 0,14 0,053 0,33	0,084 0,38 0,32 0,17 0,15 0,21	0,44 -0,0055 1.	

da información sobre esto. En esta suma, el sumando devuelve una entrada positiva para el vector de datos p-ésimo exactamente cuando las desviaciones de los componentes i-ésimo y j-ésimo el promedio ambos tienen el mismo signo. Si tienen signos diferentes, entonces la entrada es negativo. Por lo tanto, la covarianza $\sigma_{12,13}$ de los dos valores de fiebre diferentes debe ser bastante claramente positivo.

Sin embargo, la covarianza también depende del valor absoluto de las variables, lo que dificulta la comparación de los valores. Para poder comparar el grado de dependencia en el caso de múltiples variables, por lo tanto definimos la correlación coeficiente

$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}$$

para dos valores x_i y x_j , que no es más que una covarianza normalizada. La matriz K de todos los coeficientes de correlación contiene valores entre -1 y 1 , es simétrica, y todos sus elementos diagonales tienen el valor 1. La matriz de correlación para los 16 variables se da en la Tabla 8.3.

Esta matriz se vuelve algo más legible cuando la representamos como una densidad trama. En lugar de los valores numéricos, los elementos de la matriz en la Fig. 8.6 en la página 169 se rellenan con valores grises. En el diagrama de la derecha, se muestran los valores absolutos. De este modo podemos ver muy rápidamente qué variables muestran una dependencia débil o fuerte. Nosotros puede ver, por ejemplo, que las variables 7, 9, 10 y 14 muestran la correlación más fuerte con la variable de clase apendicitis y por lo tanto son más importantes para el diagnóstico que la otra variable. También vemos, sin embargo, que las variables 9 y 10 están fuertemente correlacionados. Esto podría significar que uno de estos dos valores es potencialmente suficiente para el diagnóstico.

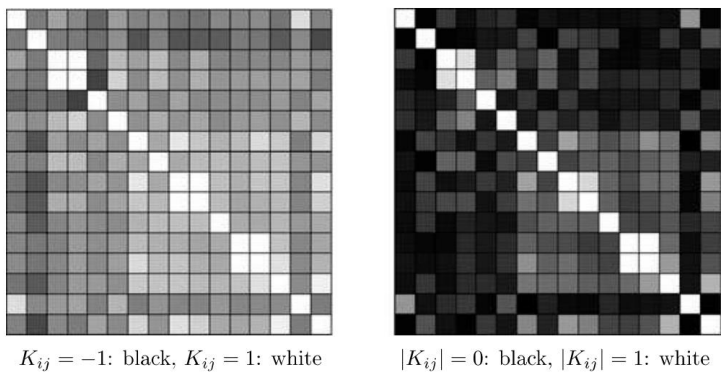
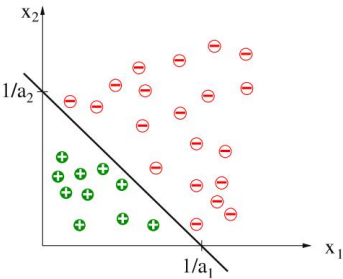


Fig. 8.6 La matriz de correlación como gráfico de frecuencia. En el diagrama de la izquierda, oscuro representa negativo y claro positivo. En la imagen de la derecha se listaron los valores absolutos. Aquí negro significa $K_{ij} \approx 0$ (sin correlación) y blanco $|K_{ij}| \approx 1$ (fuertemente correlacionado)

Fig. 8.7 Un conjunto de datos bidimensional linealmente separable. La ecuación de la recta divisoria es $a_1x_1 + a_2x_2 = 1$



8.2 El perceptrón, un clasificador lineal

En el ejemplo de clasificación de clasificación de manzanas, se dibuja una línea divisoria curva entre las dos clases en la Fig. 8.3 en la página 163. En la Fig. 8.7 se muestra un caso más simple . Aquí los ejemplos de entrenamiento bidimensionales pueden estar separados por una línea recta. Llamamos a este conjunto de datos de entrenamiento linealmente separable. En n dimensiones se necesita un hiperplano para la separación. Esto representa un subespacio lineal de dimensión $n - 1$.

Debido a que todo hiperplano de dimensión $(n - 1)$ en la ecuación $\dots$ puede ser descrito por un de R

$$ax_i = \theta$$

you 1

tiene sentido definir la separabilidad lineal de la siguiente manera.

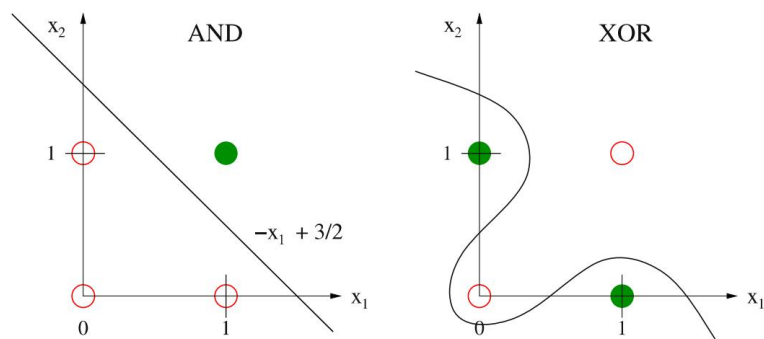


Fig. 8.8 La función booleana AND es linealmente separable, pero XOR no lo es ($\bullet$ = verdadero, $\circ$ = falso)

Definición 8.2 Dos conjuntos $M1 \subseteq \mathbb{R}^n$ y $M2 \subseteq \mathbb{R}^n$ se llaman linealmente separables si los números reales $a_1, \dots, a_n, \theta$ existen con

$$a_i x_i > \theta \text{ para todo } x \in M1$$

$$a_i x_i \leq \theta \text{ para todo } x \in M2.$$

El valor θ se denota como el umbral.

En la figura 8.8 vemos que la función AND es linealmente separable, pero la función XOR no lo es. Para AND, por ejemplo, la línea $-x_1 + 3/2$ separa las interpretaciones verdaderas y falsas de la fórmula $x_1 \wedge x_2$. Por el contrario, la función XOR no tiene una línea recta de separación. Claramente, la función XOR tiene una estructura más compleja que la función AND en este sentido.

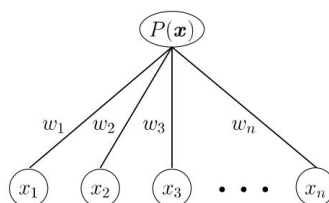
Con el perceptrón, presentamos un algoritmo de aprendizaje muy simple que puede separar ara conjuntos linealmente separables.

Definición 8.3 Sea $w = (w_1, \dots, w_n) \in \mathbb{R}^n$ un vector de peso y $x \in \mathbb{R}^n$ sea un vector de entrada. Un perceptrón representa una función $P : \mathbb{R}^n \rightarrow \{0, 1\}$ que a la siguiente regla:

$$P(x) = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i > 0, \\ 0 & \text{más} \end{cases}$$

El perceptrón [Ros58, MP69] es un algoritmo de clasificación muy simple. Es equivalente a una red neuronal de dos capas con activación por una función de umbral, que se muestra en la Fig. 8.9 en la página 171. Como se muestra en el Cap. 9, cada nodo de la red representa una neurona y cada borde una sinapsis. Por ahora, sin embargo, sólo vamos a

Fig. 8.9 Representación gráfica de un perceptrón como una red neuronal de dos capas



ver el perceptrón como un agente de aprendizaje, es decir, como una función matemática que asigna un vector de características a un valor de función. Aquí las variables de entrada x_i se denotan características.

Como podemos ver en la fórmula $\sum_{i=1}^n w_i x_i > 0$, todos los puntos x sobre el hiperplano $\sum_{i=1}^n w_i x_i = 0$ se clasifican como positivos ($P(x) = 1$), y todos los demás como negativos ($P(x) = 0$). El hiperplano de separación pasa por el origen porque $\theta = 0$.

Usaremos un pequeño truco para mostrar que la ausencia de un umbral arbitrario no representa ninguna restricción de poder. Primero, sin embargo, queremos presentar un algoritmo de aprendizaje simple para el perceptrón.

8.2.1 La regla de aprendizaje

Con la notación M^+ y M^- para los conjuntos de patrones de entrenamiento positivos y negativos respectivamente, la regla de aprendizaje del perceptrón dice [MP69]

```

APRENDIZAJE PERCEPTRON [ $M^+, M^-$ ]  $w =$ 
vector arbitrario de números reales
Repetir
  Para todo  $x \in M^+$ 
    Si  $w \cdot x \leq 0$  Entonces  $w = w + x$ 
  Para todo  $x \in M^-$ 
    Si  $w \cdot x > 0$  Entonces  $w = w - x$ 
Hasta que todo  $x \in M^+ \cup M^-$  se clasifique correctamente
  
```

El perceptrón debe generar el valor 1 para todo $x \in M^+$. Según la Definición 8.3 en la página 170, esto es cierto cuando $w \cdot x > 0$. Si este no es el caso, entonces se agrega x al vector de peso w , por lo que el vector de peso cambia exactamente en la dirección correcta.

Vemos esto cuando aplicamos el perceptrón al vector cambiado $w + x$ porque

$$2(w + x) \cdot x = w \cdot x + x \cdot x$$

Si este paso se repite con suficiente frecuencia, en algún momento el valor $w \cdot x$ se volverá positivo, como debería ser. Análogamente, vemos que, para datos de entrenamiento negativos, la

perceptron calcula un valor cada vez más pequeño

$$(w - x) \cdot x = wx - x^2$$

que en algún momento se vuelve negativa.²

Ejemplo 8.2 Se va a entrenar un perceptrón en los conjuntos $M^+ = \{(0, 1.8), (2, 0.6)\}$ y $M^- = \{(-1.2, 1.4), (0.4, -1)\}$. $w = (1, 1)$ se utilizó como vector de peso inicial. Los datos de entrenamiento y la línea definida por el vector de peso $wx = x_1 + x_2 = 0$ se muestran en la Fig. 8.10 en la página 173 en la primera imagen de la fila superior. Además, el vector de peso se dibuja como una línea discontinua. Como $wx = 0$, esto es ortogonal a la línea.

En la primera iteración a través del ciclo del algoritmo de aprendizaje, el único falso El ejemplo de entrenamiento clasificado es $(-1.2, 1.4)$ porque

$$(-1.2, 1.4) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0.2 > 0.$$

Esto da como resultado $w = (1, 1) - (-1.2, 1.4) = (2.2, -0.4)$, como se dibuja en la segunda imagen en la fila superior de la Fig. 8.10 en la página 173. Las otras imágenes muestran cómo, después de un total de cinco cambios, la línea divisoria se encuentra entre las dos clases. El perceptrón clasifica así todos los datos correctamente. Vemos claramente en el ejemplo que cada punto de datos clasificado incorrectamente de M^+ "jala" el vector de peso w en su dirección y cada punto clasificado incorrectamente de M^- "empuja" el vector de peso en la dirección opuesta.

Se ha demostrado [MP69] que el perceptrón siempre converge para sep linealmente datos cultivables. Tenemos

Teorema 8.1 Sean las clases M^+ y M^- linealmente separables por un hiperplano $wx = 0$. Entonces PERCEPTRONLEARNING converge para cada inicialización del vector w . El perceptrón P con el vector de peso así calculado divide las clases M^+ y M^- , es decir:

$$\begin{aligned} \text{PAG}(x) &= 1 & x &\in M^+ \\ y \\ \text{PAGS}(x) &= 0 & x &\in \text{METRO}^-. \end{aligned}$$

Como podemos ver claramente en el ejemplo 8.2, los perceptrones definidos anteriormente no pueden dividir conjuntos separables linealmente arbitrarios, sino solo aquellos que son divisibles por una línea que pasa por el origen, o en $\mathbb{R}^n$ por un hiperplano que pasa por el origen, porque el término constante θ es falta en la ecuación $\sum_{i=1}^n w_i x_i = 0$.

²[Precaución! Esta no es una prueba de convergencia para la regla de aprendizaje del perceptrón. Solo muestra que el perceptrón converge cuando el conjunto de datos de entrenamiento consiste en un solo ejemplo.

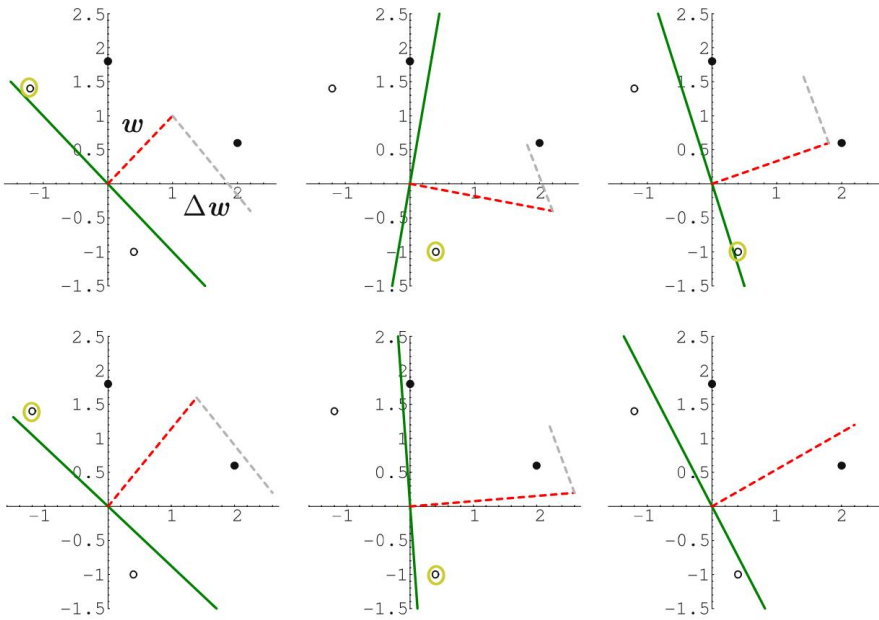


Fig. 8.10 Aplicación de la regla de aprendizaje del perceptrón a dos puntos de datos positivos (•) y dos negativos (◦). La línea continua muestra la línea divisoria actual $wx = 0$. La línea discontinua ortogonal es el vector de peso w y la segunda línea discontinua el vector de cambio $w = x$ o $w = -x$ que se agregará, que se calcula a partir del punto de datos actualmente activo rodeado de gris

Con el siguiente truco podemos generar el término constante. Mantenemos constante el último componente x_n del vector de entrada x y lo establecemos en el valor 1. Ahora el peso $w_n =: -\theta$ funciona como un umbral porque

$$\begin{array}{ccc} \text{--} & n-1 & n-1 \\ wix_i = & wix_i - \theta > 0 & wix_i > \theta \\ y_{o=1} & y_{o=1} & y_{o=1} \end{array}$$

Tal valor constante $x_n = 1$ en la entrada se llama unidad de polarización. Debido a que el peso asociado provoca un desplazamiento constante del hiperplano, el término "sesgo" encaja bien.

En la aplicación del algoritmo de aprendizaje del perceptrón, se añade un bit con el valor constante 1 al vector de datos de entrenamiento. Observamos que el peso w_n , o el umbral θ , se aprende durante el proceso de aprendizaje.

Ahora se ha demostrado que un perceptrón $P_\theta : \mathbb{R}^{n-1} \rightarrow \{0, 1\}$

$$P_\theta(x_1, \dots, x_{n-1}) = \begin{cases} 1 & \text{si } \sum_{i=1}^{n-1} w_i x_i > \theta \\ 0 & \text{más} \end{cases} \quad (8.1)$$

con un umbral arbitrario se puede simular mediante un perceptrón $P : \mathbb{R}^n \rightarrow \{0, 1\}$ con el umbral 0. Si comparamos (8.1) con la definición de linealmente separable, vemos que ambas afirmaciones son equivalentes. En resumen, hemos demostrado que:

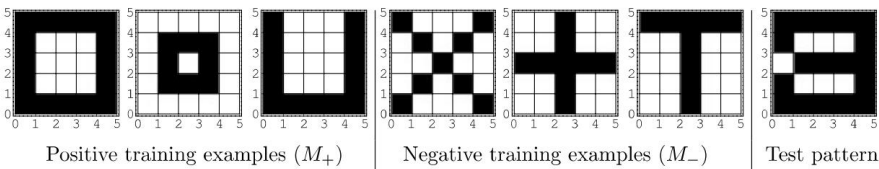
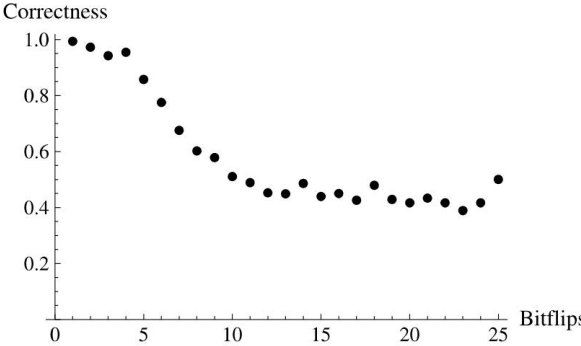


Fig. 8.11 Los seis patrones utilizados para el entrenamiento. El patrón completo de la derecha es uno de los 22 patrones de prueba para el primer patrón con una secuencia de cuatro bits invertidos

Fig. 8.12 Corrección relativa del perceptrón en función del número de bits invertidos en los datos de prueba



Teorema 8.2 Una función $f : \mathbb{R} \rightarrow \{0, 1\}$ puede representarse mediante un perceptrón si y solo si los dos conjuntos de vectores de entrada positivos y negativos son linealmente separables.

Ejemplo 8.3 Ahora entrenamos un perceptrón con un umbral en seis patrones binarios gráficos simples, representados en la figura 8.11, con 5×5 píxeles cada uno.

Los datos de entrenamiento pueden ser aprendidos por PERCEPTRONLEARNING en cuatro iteraciones sobre todos los patrones. Se utilizan patrones con un número variable de bits invertidos como ruido para probar la capacidad de generalización del sistema. Los bits invertidos en el patrón de prueba están en cada caso en secuencia uno tras otro. En la figura 8.12, el porcentaje de patrones clasificados correctamente se representa en función del número de bits falsos.

Después de unos cinco bits invertidos consecutivos, la corrección cae bruscamente, lo que no sorprende dada la simplicidad del modelo. En la siguiente sección presentaremos un algoritmo que funciona mucho mejor en este caso.

8.2.2 Optimización y Outlook

Como uno de los algoritmos de aprendizaje basados en redes neuronales más simples, el perceptrón de dos capas solo puede dividir clases linealmente separables. Insecto. 9.5 veremos que las redes multicapa son significativamente más poderosas. A pesar de su estructura simple, el

perceptron en la forma presentada converge muy lentamente. Puede acelerarse mediante la normalización del vector de alteración del peso. Las fórmulas $w = w \pm x$ se sustituyen por $w = w \pm x/|x|$. Por lo tanto, cada punto de datos tiene el mismo peso durante el aprendizaje, independientemente de su valor.

La velocidad de convergencia depende en gran medida de la inicialización del vector w . Idealmente, no sería necesario cambiarlo en absoluto y el algoritmo convergería después de una iteración. Podemos acercarnos a este objetivo usando la inicialización heurística

$$w_0 = \frac{x^- - x^+}{x^+ - x^-},$$

que investigaremos más de cerca en el ejercicio 8.5 de la página 217.

Si comparamos la fórmula del perceptrón con el método de puntuación presentado en la Secc. 7.3.1, vemos inmediatamente su equivalencia. Además, el perceptrón, como el modelo de red neuronal más simple, es equivalente a Naive Bayes, el tipo más simple de red bayesiana (vea el Ejercicio 8.16 en la página 219). Así, evidentemente, varios algoritmos de clasificación muy diferentes tienen un origen común.

En el cap. 9 nos familiarizaremos con una generalización del perceptrón en forma de algoritmo de retropropagación, que puede dividir conjuntos separables no linealmente mediante el uso de múltiples capas, y que posee una mejor regla de aprendizaje.

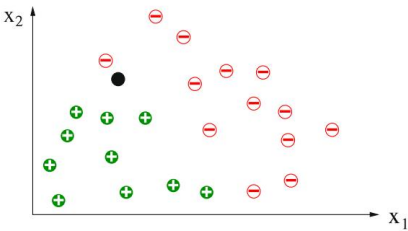
8.3 El método del vecino más cercano

Para un perceptrón, el conocimiento disponible en los datos de entrenamiento se extrae y se guarda de forma comprimida en los pesos w_i . Por lo tanto, se pierde información sobre los datos. Esto es exactamente lo que se desea, sin embargo, si se supone que el sistema debe generalizar los datos de entrenamiento a nuevos datos. En este caso, la generalización es un proceso que requiere mucho tiempo y cuyo objetivo es encontrar una representación compacta de los datos en forma de una función que clasifique los nuevos datos de la mejor manera posible.

La memorización de todos los datos simplemente guardándolos es bastante diferente. Aquí el aprendizaje es extremadamente simple. Sin embargo, como se mencionó anteriormente, el conocimiento guardado no es tan fácil de aplicar a ejemplos nuevos y desconocidos. Tal enfoque es muy inadecuado para aprender a esquiar, por ejemplo. Un principiante nunca puede convertirse en un buen esquiador simplemente viendo videos de buenos esquiadores. Evidentemente, cuando se realizan automáticamente movimientos de aprendizaje de este tipo, ocurre algo similar al caso del perceptrón. Después de una práctica suficientemente larga, el conocimiento almacenado en los ejemplos de entrenamiento se transforma en una representación interna en el cerebro.

Sin embargo, hay ejemplos exitosos de memorización en los que también es posible la generalización. Durante el diagnóstico de un caso difícil, un médico podría tratar de recordar casos similares del pasado. Si su memoria es buena, entonces podría dar con este caso, buscarlo en sus archivos y finalmente llegar a un diagnóstico similar. Para este enfoque, el médico debe tener un buen sentido de la similitud, a fin de recordar el caso más similar. Si ha encontrado esto, entonces debe preguntarse si es lo suficientemente similar para justificar el mismo diagnóstico.

Fig. 8.13 En este ejemplo con ejemplos de entrenamiento negativos y positivos, el método del vecino más cercano agrupa el nuevo punto marcado en negro en la clase negativa



¿Qué significa semejanza en el contexto formal que estamos construyendo? Representamos las muestras de entrenamiento como de costumbre en un espacio de características multidimensionales y definimos: Cuanto menor sea su distancia en el espacio de características, más similares serán dos ejemplos. Ahora aplicamos esta definición al ejemplo bidimensional simple de la figura 8.13. Aquí el próximo vecino al punto negro es un ejemplo negativo. Por lo tanto, se asigna a la clase negativa.

La distancia $d(x, y)$ entre dos puntos x y y medida por R^m puede ser por ejemplo la distancia euclidiana

$$d(x, y) = |x - y| = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Debido a que existen muchas otras métricas de distancia además de esta, tiene sentido pensar en alternativas para una aplicación concreta. En muchas aplicaciones, ciertas características son más importantes que otras. Por lo tanto, a menudo es sensato escalar las características de manera diferente por pesos w_i . La fórmula entonces dice

$$dw(x, y) = |x - y| = \sqrt{\sum_{i=1}^m w_i (x_i - y_i)^2}$$

El siguiente programa simple de clasificación de vecinos más cercanos busca en los datos de entrenamiento el vecino más cercano t para el nuevo ejemplo s y luego clasifica s exactamente como t .

```
VECINO MÁS PRÓXIMO[M+,M-,s] t =
  argminx M+ M- {d(s, x)}
  Si t M+ Entonces Vuelve („+")
  De lo contrario Retorno („-")
```

3Los funcionales argmin y argmax determinan, al igual que min y max, el mínimo o máximo de un conjunto o función. Sin embargo, en lugar de devolver el valor del máximo o mínimo, dan la posición, es decir, el argumento en el que aparece el extremo.

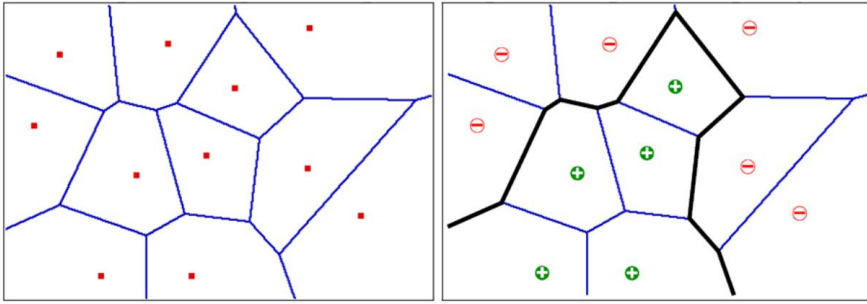
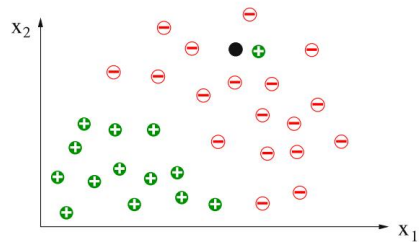


Fig. 8.14 Un conjunto de puntos junto con su diagrama de Voronoi (izquierda) y la línea divisoria generada para las dos clases M+ y M-

Fig. 8.15 El método del vecino más cercano asigna el nuevo punto marcado en negro a la clase incorrecta (positiva) porque lo más probable es que el vecino más cercano se clasifique incorrectamente



A diferencia del perceptrón, el método del vecino más cercano no genera una línea que divida los puntos de datos de entrenamiento. Sin embargo, ciertamente existe una línea imaginaria que separa las dos clases. Podemos generar esto generando primero el llamado diagrama de Voronoi. En el diagrama de Voronoi, cada punto de datos está rodeado por un polígono convexo, que define una vecindad a su alrededor. El diagrama de Voronoi tiene la propiedad de que para un nuevo punto arbitrario, el vecino más cercano entre los puntos de datos es el punto de datos, que se encuentra en la misma vecindad. Si se determina el diagrama de Voronoi para un conjunto de datos de entrenamiento, entonces es sencillo encontrar el vecino más cercano para un nuevo punto que se va a clasificar. La membresía de la clase se tomará del vecino más cercano.

En la figura 8.14 vemos claramente que el método del vecino más cercano es significativamente más poderoso que el perceptrón. Es capaz de realizar correctamente líneas divisorias arbitrariamente complejas (en general: hiperplanos). Sin embargo, aquí hay un peligro. Un solo punto erróneo puede, en determinadas circunstancias, conducir a resultados de clasificación muy malos. Tal caso ocurre en la Fig. 8.15 durante la clasificación del punto negro. El método del vecino más cercano puede clasificar esto incorrectamente. Si el punto negro está inmediatamente al lado de un punto positivo que es un valor atípico de la clase positiva, entonces se clasificará como positivo en lugar de negativo, como se pretende aquí. Un ajuste erróneo a errores aleatorios (ruido) se denomina sobreajuste.

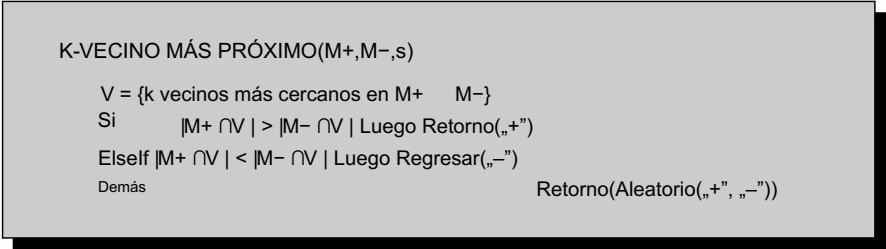
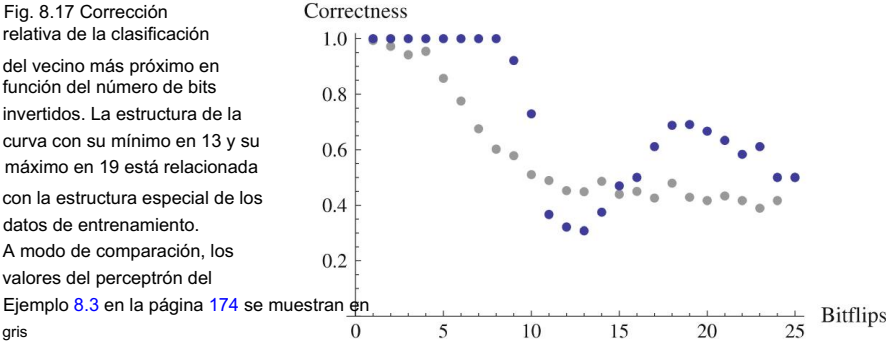


Fig. 8.16 El ALGORITMO K-VECINO MÁS PRÓXIMO



Para evitar clasificaciones falsas debido a valores atípicos únicos, se recomienda suavizar un poco la superficie de división. Esto se puede lograr, por ejemplo, con el algoritmo K-VECINO MÁS CERCANO de la figura 8.16, que toma una decisión mayoritaria entre los k vecinos más cercanos.

Ejemplo 8.4 Ahora aplicamos el VECINO MÁS PRÓXIMO al Ejemplo 8.3 de la página 174. Debido a que estamos tratando con datos binarios, usamos la distancia de Hamming como la métrica de distancia.⁴ Como ejemplo de prueba, nuevamente usamos ejemplos de entrenamiento modificados con n bits invertidos consecutivos cada uno. En la Fig. 8.17 se muestra el porcentaje de ejemplos de prueba correctamente clasificados en función del número de bits invertidos b. Para hasta ocho bits invertidos, todos los patrones se identifican correctamente. Más allá de ese punto, el número de errores aumenta rápidamente. Esto no sorprende porque el patrón de entrenamiento número 2 de la Fig. 8.11 en la página 174 de la clase M+ tiene una distancia de hamming de 9 con respecto a los dos ejemplos de entrenamiento, los números 4 y 5 de la otra clase. Esto significa que es muy probable que el patrón de prueba esté cerca de los patrones de la otra clase. Claramente vemos que la clasificación del vecino más cercano es superior al perceptrón en esta aplicación para hasta ocho bits falsos.

⁴La distancia de Hamming entre dos vectores de bits es el número de bits diferentes de los dos vectores.

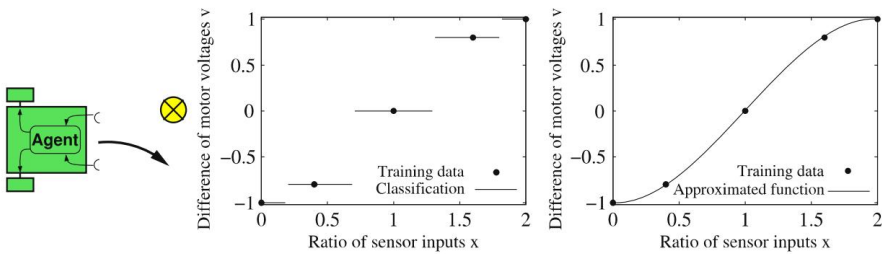


Fig. 8.18 El agente de aprendizaje, que se supone que debe evitar la luz (izquierda), representado como un clasificador (centro) y como una aproximación (derecha)

8.3.1 Dos clases, muchas clases, aproximación

La clasificación del vecino más cercano también se puede aplicar a más de dos clases. Al igual que en el caso de dos clases, la clase del vector de características a clasificar simplemente se establece como la clase del vecino más cercano. Para el método del k vecino más cercano, la clase debe determinarse como la clase con la mayor cantidad de miembros entre los k vecinos más cercanos.

Si el número de clases es grande, normalmente ya no tiene sentido utilizar algoritmos de clasificación porque el tamaño de los datos de entrenamiento necesarios crece rápidamente con el número de clases. Además, en determinadas circunstancias, se pierde información importante durante la clasificación de muchas clases. Esto quedará claro en el siguiente ejemplo.

Ejemplo 8.5 Se supone que un robot autónomo con sensores simples similares a los vehículos Braitenberg presentados en la Fig. 1.1 en la página 2 debe aprender a alejarse de la luz. Esto significa que debe aprender de la manera más óptima posible a mapear los valores de sus sensores en una señal de dirección que controla la dirección de conducción. El robot está equipado con dos sensores de luz simples en su parte frontal. A partir de las dos señales del sensor (con sl para el sensor izquierdo y sr para el sensor derecho), se calcula la relación $x = sr/sl$. Para controlar los motores eléctricos de las dos ruedas a partir de este valor x , la diferencia $v = Ur - Ul$ de los dos voltajes Ur y Ul de los motores izquierdo y derecho, respectivamente. La tarea del agente de aprendizaje es ahora evitar una señal luminosa. Por lo tanto, debe aprender un mapeo f que calcula el valor "correcto" $v = f(x)$.

Para ello llevamos a cabo un experimento en el que, para unos pocos valores x medidos, encontramos un valor v tan óptimo como podamos. Estos valores se trazan como puntos de datos en la figura 8.18 y servirán como datos de entrenamiento para el agente de aprendizaje. Durante la clasificación del vecino más cercano, cada punto en el espacio de características (es decir, en el eje x) se clasifica exactamente como su vecino más cercano entre los datos de entrenamiento. La función para dirigir los motores es entonces una función escalonada con grandes saltos (Fig. 8.18 centro). Si queremos pasos más finos, debemos proporcionar correspondientemente más datos de entrenamiento. En

5Para mantener el ejemplo simple y legible, el vector de características x se mantuvo deliberadamente dimensional.

por otro lado, podemos obtener una función continua si aproximamos una función suave para ajustar los cinco puntos (Fig. 8.18 en la página 179 derecha). Requerir que la función f sea continua conduce a muy buenos resultados, incluso sin puntos de datos adicionales.

Para la aproximación de funciones sobre puntos de datos existen muchos métodos matemáticos, como la interpolación polinomial, la interpolación spline o el método de los mínimos cuadrados. La aplicación de estos métodos se vuelve problemática en dimensiones superiores. La dificultad especial de la IA es que se necesitan métodos de aproximación sin modelo. Es decir, se debe realizar una buena aproximación de los datos sin conocer las propiedades especiales de los datos o la aplicación. Aquí se han logrado muy buenos resultados con redes neuronales y otros aproximadores de funciones no lineales, que se presentan en el Cap. 9.

El método del k vecino más próximo se puede aplicar de forma sencilla al problema de aproximación. En el algoritmo K-VECINO MÁS CERCANO, después de que se determina el conjunto $V = \{x_1, x_2, \dots, x_k\}$, el valor de la función promedio de k vecinos más cercanos

$$f(x) = \frac{1}{k} \sum_{i=1}^k f(x_i) \quad (8.2)$$

se calcula y se toma como una aproximación $\hat{f}$ para el vector de consulta x . Cuanto mayor sea k , más suave será la función $\hat{f}$.

8.3.2 La distancia es relevante

En la aplicación práctica de variantes discretas y continuas del método del k vecino más cercano, a menudo surgen problemas. A medida que k se vuelve grande, normalmente existen más vecinos con una distancia grande que aquellos con una distancia pequeña. Por tanto, el cálculo de $\hat{f}$ está dominado por los vecinos que están lejos. Para evitar esto, los k vecinos se ponderan de manera que los vecinos más distantes tengan menor influencia en el resultado. Durante la decisión mayoritaria en el algoritmo K-VECINO MÁS CERCANO, los "votos" se ponderan con el peso

$$w_i = \frac{1}{1 + \alpha d(x, x_i)^2}, \quad (8.3)$$

que decrece con el cuadrado de la distancia. La constante α determina la velocidad de disminución de los pesos. La ecuación (8.2) ahora se reemplaza por

$$f(x) = \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}.$$

Para una concentración uniformemente distribuida de puntos en el espacio de características, esto asegura que la influencia de los puntos se acerque asintóticamente a cero a medida que aumenta la distancia. Por lo tanto, es posible utilizar muchos o incluso todos los datos de entrenamiento para clasificar o aproximar un vector de entrada dado.

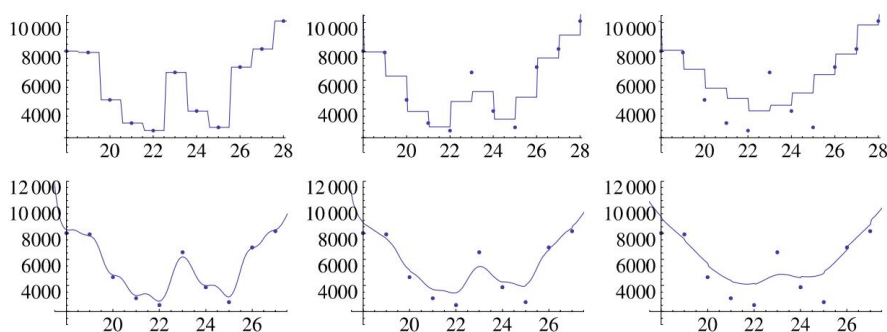


Fig. 8.19 Comparación del método del k vecino más cercano (fila superior) con $k = 1$ (izquierda), $k = 2$ (medio) y $k = 6$ (derecha), con su variante ponderada por distancia (fila inferior) con $\alpha = 20$ (izquierda), $\alpha = 4$ (centro) y $\alpha = 1$ (derecha) en un conjunto de datos unidimensional

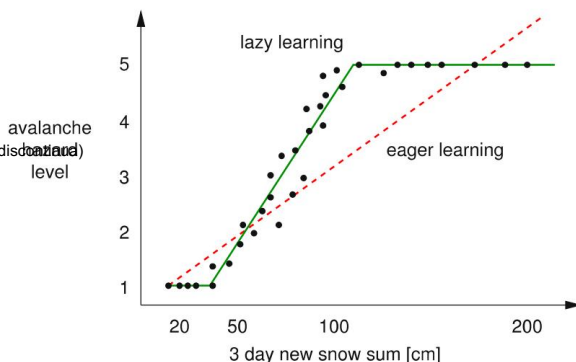
Para tener una idea de estos métodos, en la figura 8.19 se compara el método del k vecino más cercano (en la fila superior) con su optimización ponderada por la distancia. Debido al promedio, ambos métodos pueden generalizar, o en otras palabras, cancelar el ruido, si el número de vecinos para k -vecino más cercano o el parámetro α se configura adecuadamente. Los diagramas muestran muy bien que el método de distancia ponderada proporciona una aproximación mucho más suave que el k -vecino más cercano. Con respecto a la calidad de la aproximación, este método tan simple puede competir bien con algoritmos de aproximación sofisticados, como redes neuronales no lineales, máquinas de vectores de soporte y procesos gaussianos.

Hay muchas alternativas a la función de peso (también llamada núcleo) dada en (8.3) en la página 180. Por ejemplo, se puede usar una función gaussiana o similar en forma de campana. Para la mayoría de las aplicaciones, los resultados no son muy sensibles a la selección del kernel. Sin embargo, el parámetro de ancho α , que para todas estas funciones debe configurarse manualmente, tiene una gran influencia en los resultados, como se muestra en la Fig. 8.19. Para evitar una adaptación manual tan inconveniente, se han desarrollado métodos de optimización para configurar automáticamente este parámetro [SA94, SE10].

8.3.3 Tiempos de Cómputo

Como se mencionó anteriormente, el entrenamiento se logra en todas las variantes del método del vecino más cercano simplemente guardando todos los vectores de entrenamiento junto con sus etiquetas (valores de clase) o el valor de la función $f(x)$. Por lo tanto, no hay otro algoritmo de aprendizaje que aprenda tan rápido. Sin embargo, responder a una consulta de clasificación o aproximación de un vector x puede resultar muy costoso. Solo encontrar los k vecinos más cercanos para n datos de entrenamiento requiere un costo que crece linealmente con n . Por clasificación o aproximación, existe adicionalmente un costo que es lineal en k . El tiempo total de cálculo crece así como $O(n + k)$. Para grandes cantidades de datos de entrenamiento, esto puede generar problemas.

Fig. 8.20 Para determinar el peligro de avalancha, se aproxima una función a partir de datos de entrenamiento. Aquí, a modo de comparación, hay un modelo local (línea continua) y un modelo global (línea discontinua).



8.3.4 Resumen y perspectiva

Debido a que no sucede nada en la fase de aprendizaje de los métodos del vecino más cercano presentados, estos algoritmos también se denominan aprendizaje perezoso, en contraste con el aprendizaje ansioso, en el que la fase de aprendizaje puede ser costosa, pero la aplicación a nuevos ejemplos es muy eficiente. El perceptrón y todas las demás redes neuronales, el aprendizaje de árboles de decisión y el aprendizaje de redes bayesianas son métodos de aprendizaje ávidos. Dado que los métodos de aprendizaje perezoso necesitan acceso a la memoria con todos los datos de entrenamiento para aproximar una nueva entrada, también se denominan aprendizaje basado en la memoria.

Para comparar estas dos clases de procesos de aprendizaje, usaremos como ejemplo la tarea de determinar el peligro de avalancha actual a partir de la cantidad de nieve recién caída en un área determinada de Suiza.⁶ En la Fig. 8.20 se ingresan valores determinados por expertos, que queremos usar como datos de entrenamiento. Durante la aplicación de un algoritmo de aprendizaje entusiasta que realiza una aproximación lineal de los datos, se calcula la línea discontinua que se muestra en la figura. Debido a la restricción a una línea recta, el error es relativamente grande con un máximo de alrededor de 1,5 niveles de riesgo. Durante el aprendizaje diferido, no se calcula nada antes de que llegue una consulta sobre el nivel de riesgo actual.

Luego, la respuesta se calcula a partir de varios vecinos más cercanos, es decir, localmente. Podría dar como resultado la curva que se muestra en la figura, que se forma a partir de segmentos de línea y muestra errores mucho más pequeños. La ventaja del método perezoso es su localidad. La aproximación se toma localmente del nuevo nivel de nieve actual y no globalmente.

Por lo tanto, para clases de funciones fundamentalmente iguales (por ejemplo, funciones lineales), los algoritmos perezosos son mejores.

Los métodos del vecino más cercano son adecuados para todas las situaciones problemáticas en las que se necesita una buena aproximación local, pero que no exigen mucho de la velocidad del sistema. El predictor de avalancha mencionado aquí, que se ejecuta una vez al día, es una aplicación de este tipo. Los métodos del vecino más cercano no son adecuados cuando una descripción del conocimiento extraído de los datos debe ser comprensible para los humanos, lo que hoy en día es el caso de muchas aplicaciones de minería de datos (consulte la Sección 8.4). En

⁶El total de tres días de nevadas es, de hecho, una característica importante para determinar el nivel de peligro. En la práctica, sin embargo, se utilizan atributos adicionales [Bra01]. El ejemplo utilizado aquí está simplificado.

Característica	Consulta	Caja desde la base de la caja
Pieza defectuosa:	Luz trasera	Luz delantera
Modelo de bicicleta:	Montaña Pino Marín	VSF T400
Año:	1993	2001
Fuente de alimentación:	Batería	Dinamo
Estado de la bombilla:	bien	OK
Estado del cable de luz:	?	OK
Solución		
Diagnóstico:	?	Falta el contacto eléctrico delantero
Reparar:	?	Establecer contacto eléctrico frontal

Fig. 8.21 Ejemplo de diagnóstico simple para una consulta y el caso correspondiente de la base de casos

En los últimos años, estos métodos de aprendizaje basados en la memoria se están volviendo populares y se han introducido varias variantes mejoradas (por ejemplo, la regresión lineal ponderada localmente). aplicado [Cle79].

Para poder utilizar los métodos descritos, los datos de entrenamiento deben estar disponibles en forma de vectores de números enteros o números reales. Por lo tanto, no son adecuados para aplicaciones en las que los datos se representan simbólicamente, por ejemplo como primera fórmulas lógicas de orden. Ahora hablaremos brevemente de esto.

8.3.5 Razonamiento basado en casos

En el razonamiento basado en casos (CBR), el método del vecino más cercano se extiende a las descripciones de problemas simbólicos y sus soluciones. CBR se utiliza en el diagnóstico de problemas técnicos en atención al cliente o para líneas telefónicas de atención. El ejemplo mostrado en la Fig. 8.21 sobre el diagnóstico de la luz de una bicicleta que se apaga ilustra este tipo de situación.

Se busca solución a la consulta de un cliente con una bicicleta trasera defectuosa luz. En la columna de la derecha, se da un caso similar a la consulta en la columna del medio. Esto se deriva de la base de casos, que corresponde a datos de entrenamiento en el más cercano método vecino. Si simplemente tomáramos el caso más similar, como hacemos en el más cercano método vecino, entonces terminaríamos tratando de reparar la luz delantera cuando el la luz trasera está rota. Por lo tanto, necesitamos una transformación inversa de la solución al problema similar descubierto de vuelta a la consulta. Los pasos más importantes en la solución. a un caso CBR se llevan a cabo en la Fig. 8.22 en la página 184. La transformación en este El ejemplo es simple: la luz trasera se asigna a la luz delantera.

Tan hermoso y simple como este método parece en teoría, en la práctica la construcción de sistemas de diagnóstico CBR es una tarea muy difícil. Las tres principales dificultades son:

Modelado Los dominios de la aplicación deben ser modelados en un contexto formal.

Aquí la monotonía lógica, que conocemos por el cap. 4, presenta dificultades. Poder el desarrollador predice y mapea todos los posibles casos especiales y variantes del problema?

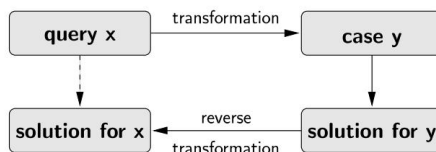


Fig. 8.22 Si para un caso x se encuentra un caso similar y , entonces para obtener una solución para x , se debe determinar la transformación y aplicar su inversa al caso descubierto y

Similitud Encontrar una métrica de similitud adecuada para características simbólicas no numéricas

Transformación Incluso si se encuentra un caso similar, aún no está claro cómo deberían verse el mapeo de transformación y su inversa.

De hecho, hoy en día se utilizan sistemas CBR prácticos para aplicaciones de diagnóstico. Sin embargo, debido a las razones mencionadas, estos se mantienen muy por detrás de los expertos humanos en rendimiento y flexibilidad. Una alternativa interesante a CBR son las redes bayesianas presentadas en el Cap. 7. A menudo, la representación simbólica del problema también se puede asignar bastante bien a características numéricas discretas o continuas. Luego, los métodos de aprendizaje inductivos mencionados, como los árboles de decisión o las redes neuronales, se pueden utilizar con éxito.

8.4 Aprendizaje del árbol de decisiones

El aprendizaje del árbol de decisiones es un algoritmo extraordinariamente importante para la IA porque es muy poderoso, pero también simple y eficiente para extraer conocimiento de los datos. En comparación con los dos algoritmos de aprendizaje ya introducidos, tiene una ventaja importante. El conocimiento extraído no solo está disponible y se puede utilizar como una función de caja negra, sino que los humanos pueden entenderlo, interpretarlo y controlarlo fácilmente en forma de un árbol de decisiones legible. Esto también hace que el aprendizaje del árbol de decisiones sea una herramienta importante para la minería de datos.

Discutiremos la función y la aplicación del aprendizaje del árbol de decisión utilizando el algoritmo C4.5. C4.5 fue introducido en 1993 por el australiano Ross Quinlan y es una mejora de su predecesor ID3 (Iterative Dichotomiser 3, 1986). Está disponible gratuitamente para uso no comercial [Qui93]. Otro desarrollo, que funciona aún más eficientemente y puede tener en cuenta los costes de las decisiones, es C5.0 [Qui93].

El sistema CART (Classification and Regression Trees, 1984) desarrollado por Leo Breiman [BFOS84] funciona de manera similar a C4.5. Tiene una interfaz gráfica de usuario conveniente, pero es muy costosa.

Veinte años antes, en 1964, J. Sonquist y J. Morgan introdujeron el sistema CHAID (Chi-square Automatic Interaction Detectors), que puede generar automáticamente árboles de decisión. Tiene la característica destacable de que detiene el crecimiento del árbol antes de que sea demasiado grande, pero hoy en día no tiene mayor relevancia.

Tabla 8.4 Variables para el problema de clasificación de esquí

Variable	Valor	Descripción
Esquí (variable objetivo)	Sí No	¿Debo conducir hasta la estación de esquí más cercana? con suficiente nieve?
Sol (característica)	Sí No	¿Hay sol hoy?
Snow_Dist (característica)	≤100, >100	Distancia a la estación de esquí más cercana con buenas condiciones de nieve (más/menos de 100 kilómetros)
Fin de semana (característica)	Sí No	¿Hoy es fin de semana?

También es interesante la herramienta de minería de datos KNIME (Konstanz Information Miner), que tiene una interfaz de usuario amigable y, utilizando la biblioteca WEKA Java, también hace posibilidad de inducción de árboles de decisión. Insecto. 8.8 presentaremos KNIME.

Ahora mostramos primero en un ejemplo simple cómo se puede construir un árbol de decisión a partir de datos de entrenamiento, para luego analizar el algoritmo y aplicarlo a los más Ejemplo complejo de LEXMED para diagnóstico médico.

8.4.1 Un ejemplo sencillo

Un esquiador devoto que vive cerca de high sierra, una hermosa cadena montañosa en California, quiere un árbol de decisiones que lo ayude a decidir si vale la pena conducir su coche a una estación de esquí en las montañas. Por lo tanto, tenemos un problema de dos clases de esquí sí/no con base en las variables enumeradas en la Tabla 8.4.

La Figura 8.23 en la página 186 muestra un árbol de decisiones para este problema. Un árbol de decisión es un árbol cuyos nodos internos representan características (atributos). Cada borde representa un valor de atributo. En cada nodo hoja se da un valor de clase.

Los datos utilizados para la construcción del árbol de decisión se muestran en la Tabla 8.5 en página 186. Cada fila de la tabla contiene los datos de un día y, como tal, representa un muestra. Tras un examen más detenido, vemos que la fila 6 y la fila 7 se contradicen entre sí. Por lo tanto, ningún algoritmo de clasificación determinista puede clasificar correctamente todos los datos. Por lo tanto, el número de datos clasificados falsamente debe ser ≥1. El árbol de la Fig. 8.23 en la página 186 clasifica los datos de forma óptima.

¿Cómo se crea tal árbol a partir de los datos? Para responder a esta pregunta vamos a primero nos restringimos a atributos discretos con un número finito de valores. Porque el número de atributos también es finito y cada atributo puede ocurrir como máximo una vez por ruta, hay un número finito de árboles de decisión diferentes. Un algoritmo simple y obvio para la construcción de un árbol simplemente generaría todos los árboles, luego para cada árbol calcular el número de clasificaciones erróneas de los datos, y al final elegir el árbol con el mínimo número de errores. Así tendríamos incluso un algoritmo óptimo (en el sentido de errores para los datos de entrenamiento) para el aprendizaje del árbol de decisión.

La desventaja obvia de este algoritmo es su tiempo de cálculo inaceptablemente alto, tan pronto como el número de atributos se vuelve algo mayor. Lo haremos Ahora desarrolle un algoritmo heurístico que, comenzando desde la raíz, construya recursivamente

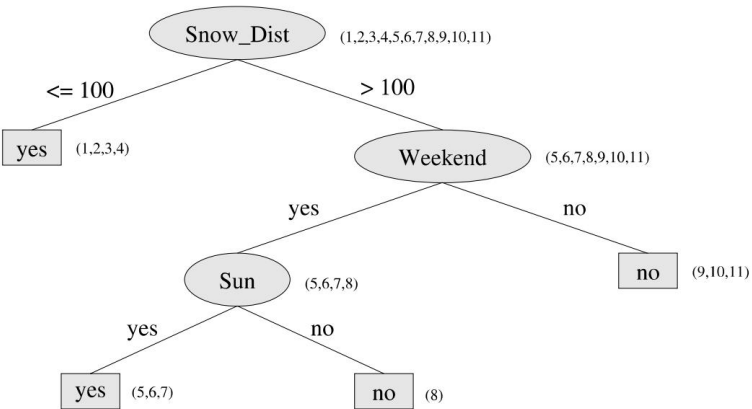


Fig. 8.23 Árbol de decisión para el problema de clasificación de esquí. En las listas a la derecha de los nodos, se dan los números de los datos de entrenamiento correspondientes. Observe que de los nodos de hoja soleado = sí solo dos de los tres ejemplos están clasificados correctamente

Tabla 8.5 Conjunto de datos para el problema de clasificación de esquí

Día	Snow_Dist	Fin de semana	Sol	Esquí
1	≤100	Sí	sí	Sí
2	≤100	Sí	sí	Sí
3	≤100	Sí	No	si
4	≤100	No	sí	Sí
5	>100	Sí	sí	Sí
6	>100	Sí	sí	Sí
7	>100	Sí	sí	No
8	>100	Sí	No	no
9	>100	No	sí	No
10	>100	No	sí	No
11	>100	No	No	no

un árbol de decisión. Primero, el atributo con la mayor ganancia de información (Snow_Dist) es elegido para el nodo raíz del conjunto de todos los atributos. Para cada valor de atributo (≤100, >100) hay una rama en el árbol. Ahora para cada rama este proceso se repite recursivamente. Durante la generación de los nodos, siempre se elige el atributo con mayor ganancia de información entre los atributos que aún no han sido utilizados, en el espíritu de una estrategia codiciosa.

8.4.2 La entropía como métrica del contenido de la información

El algoritmo descendente descrito para la construcción de un árbol de decisión, en cada paso selecciona el atributo con la mayor ganancia de información. Ahora presentamos el

entropía como la métrica para el contenido de información de un conjunto de datos de entrenamiento D. Si solo observamos la variable binaria esquiar en el ejemplo anterior, entonces D se puede describir como

$$D = (\text{sí}, \text{sí}, \text{sí}, \text{sí}, \text{sí}, \text{sí}, \text{no}, \text{no}, \text{no}, \text{no}, \text{no})$$

con probabilidades estimadas

$$p_1 = P(\text{sí}) = 6/11 \text{ y } p_2 = P(\text{no}) = 5/11.$$

Aquí evidentemente tenemos una distribución de probabilidad $p = (6/11, 5/11)$. En general, para un problema de clase n esto se lee

$$p = (p_1, \dots, p_n)$$

con

$$\sum_{i=1}^n p_i = 1.$$

Para introducir el contenido de información de una distribución observamos dos casos extremos.

Primero deja

$$p = (1, 0, 0, \dots, 0). \quad (8.4)$$

Es decir, el primero de los n eventos ciertamente ocurrirá y todos los demás no. La incertidumbre sobre el resultado de los acontecimientos es, por tanto, mínima. En cambio, para la distribución uniforme

$$p = \left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n} \right) \quad (8.5)$$

la incertidumbre es máxima porque ningún evento puede distinguirse de los demás.

Aquí Claude Shannon se preguntó cuántos bits se necesitarían para codificar tal evento. En el caso concreto de (8.4) se necesitan bits cero porque sabemos que siempre ocurre el caso $i = 1$. En el caso uniformemente distribuido de (8.5) hay n posibilidades igualmente probables. Para codificaciones binarias, aquí se necesitan $\log_2 n$ bits.

Debido a que todas las probabilidades individuales son $p_i = 1/n$, se necesitan $\log_2 n$ bits para esta codificación.

En el caso general $p = (p_1, \dots, p_n)$, si las probabilidades de los eventos elementales se desvían de la distribución uniforme, entonces se calcula el valor esperado H para el número de bits. Para ello ponderaremos todos los valores $\log_2 = -\log_2 p_i$ con sus probabilidades y obtendremos

$$H = \sum_{i=1}^n p_i (-\log_2 p_i) = - \sum_{i=1}^n p_i \log_2 p_i.$$

Cuanto más bits necesitamos para codificar un evento, claramente mayor es la incertidumbre sobre el resultado. Por lo tanto definimos:



Fig. 7.9 Algunas de las reglas LEXMED con intervalos de probabilidad. “*” significa “ y ” aquí

Estimación de las probabilidades de las reglas La estructura del gráfico de dependencia describe la estructura de las reglas aprendidas.¹¹ Las reglas aquí tienen diferentes complejidades: hay reglas que solo describen la distribución de los diagnósticos posibles (reglas a priori, por ejemplo (7.13)), reglas que describen la dependencia entre el diagnóstico y un síntoma (reglas con condiciones simples, por ejemplo (7.14)), y finalmente, reglas que describen la dependencia entre el diagnóstico y dos síntomas, como se indica en la figura 7.9 en sintaxis PIT .

$$P(\text{Diag4} = \text{inflamado}) = 0,40, P \tag{7.13}$$

$$(\text{Sono2} = \text{si} | \text{Diag4} = \text{inflamado}) = 0,43, \tag{7.14}$$

$$P(P4Q2 = \text{si} | \text{Diag4} = \text{inflamado} \quad P2Q2 = \text{si}) = 0,61. \tag{7.15}$$

Para mantener la dependencia del contexto del conocimiento guardado lo más pequeña posible, todas las reglas contienen el diagnóstico en sus condiciones y no como conclusiones. Esto es bastante similar a la construcción de muchos libros de medicina con formulaciones de la amable “Con la apendicitis solemos ver . . . ”. Como se mostró previamente en el Ejemplo 7.6 en la página 121, sin embargo, esto no presenta un problema porque, usando el bayesiano fórmula, LEXMED pone automáticamente estas reglas en la forma correcta.

Los valores numéricos de estas reglas se estiman contando su frecuencia en la base de datos. Por ejemplo, el valor en (7.14) se obtiene contando y calculando

$$\frac{|\text{Diag4} = \text{inflamado} \quad \text{Sono2} = \text{si}|}{|\text{Diag4} = \text{inflamado}|} = 0,43.$$

¹¹La diferencia entre esta y una red bayesiana es, por ejemplo, que las reglas están equipadas con intervalos de probabilidad y que solo después de aplicar el principio de máxima entropía es un único modelo de probabilidad producido.

Reglas de expertos

Debido a que la base de datos de apendicitis solo contiene pacientes que se han sometido a la operación, las reglas para el dolor abdominal inespecífico (NSAP) reciben sus valores de propuestas de expertos médicos. Las experiencias en LEXMED confirman que las reglas probabilísticas son fáciles de leer y se pueden traducir directamente al lenguaje natural. Declaraciones de expertos médicos sobre las relaciones de frecuencia de síntomas específicos y el diagnóstico, ya sea de la literatura o como resultado de una entrevista, puede por lo tanto, incorporarse a la base de reglas con poco gasto. Para modelar la incertidumbre del conocimiento experto, el uso de intervalos de probabilidad ha demostrado ser efectivo. El conocimiento experto se adquirió principalmente de los cirujanos participantes, el Dr. Rampf y Dr. Hontschik, y sus publicaciones [Hon94].

Una vez que se han creado las reglas expertas, la base de reglas está terminada. Luego se calcula el modelo de probabilidad completo con el método de máxima entropía por la sistema PIT.

Consultas de diagnóstico

Usando su modelo de probabilidad almacenado eficientemente, LEXMED calcula las probabilidades para los cuatro diagnósticos posibles en pocos segundos. Por ejemplo, asumimos la siguiente salida:

Diagnóstico	Resultados del diagnóstico de PIT		
	Apéndice inflamado	Apéndice perforado	
Probabilidad	0,24	0,16	
	Negativo	Otro	
	0.57	0.03	

Se debe tomar una decisión basada en estos cuatro valores de probabilidad para buscar uno de los cuatro tratamientos: operación, operación de emergencia, observación estacionaria u observación ambulatoria.¹² Si bien la probabilidad de un diagnóstico negativo en este caso supera a los demás, enviar al paciente a casa como sano no es una buena decisión. Podemos ver claramente que, incluso cuando las probabilidades de los diagnósticos han sido calculado, el diagnóstico aún no está terminado.

Más bien, la tarea ahora es derivar una decisión óptima de estas probabilidades. A con este fin, el usuario puede hacer que LEXMED calcule una decisión recomendada.

7.3.5 Gestión de Riesgos Utilizando la Matriz de Costos

¿Cómo se pueden traducir ahora de manera óptima las probabilidades calculadas en decisiones? Un algoritmo ingenuo asignaría una decisión a cada diagnóstico y finalmente seleccionaría la decisión que corresponda a la mayor probabilidad. Suponga que el calculado las probabilidades son 0,40 para el diagnóstico de apendicitis (inflamada o perforada), 0,55 para el diagnóstico negativo, y 0,05 para el diagnóstico otro. Un algoritmo ingenuo sería ahora elija la decisión (demasiado arriesgada) “sin operación” porque corresponde al diagnóstico con mayor probabilidad. Un mejor método consiste en comparar los costos

¹² Observación ambulatoria significa que el paciente es dado de alta para quedarse en casa.

Tabla 7.3 La matriz de costos de LEXMED junto con las probabilidades de diagnóstico computadas de un paciente

Terapia	Probabilidad de varios diagnósticos.				
	Enconado	Perforado	Negativo	Otro	
	0.25	0.15	0,55	0.05	
Operación	0	500	5800	6000	3565
Operación de emergencia	500	0	6300	6500	3915
Observador ambulante	12000	150000	0	16500	26325
Otro	3000	5000	1300	0	2215
Observador estacionario	3500	7000	400	600	2175

de los posibles errores que pueden ocurrir para cada decisión. El error se cuantifica en el forma de “coste adicional (hipotético) de la decisión actual en comparación con el óptimo”. Los valores dados contienen los costos para el hospital, para la compañía de seguros, al paciente (por ejemplo, riesgo de complicaciones posoperatorias) y a otras partes (por ejemplo, ausencia del trabajo), teniendo en cuenta las consecuencias a largo plazo. Estos costos se dan en la Tabla 7.3.

Las entradas finalmente se promedian para cada decisión, es decir, se suman tomando en cuenta sus frecuencias. Estos se enumeran en la última columna de la Tabla 7.3. Finalmente, se sugiere la decisión con el menor costo promedio de error. En la Tabla 7.3 la matriz se da junto con el vector de probabilidad calculado para un paciente (en este caso: (0.25, 0.15, 0.55, 0.05)). La última columna de la tabla contiene el resultado de los cálculos de los costes medios esperados de los errores. El valor de la operación en la primera fila se calcula como $0,25 \cdot 0 + 0,15 \cdot 500 + 0,55 \cdot 5800 + 0,05 \cdot 6000 = 3565$, un promedio ponderado de todos los costos. Las decisiones óptimas se ingresan con costos (adicionales) de 0. El sistema decide el tratamiento con el promedio mínimo costo. Por lo tanto, es un ejemplo de un agente orientado a costos.

Matriz de costos en el caso binario

Para comprender mejor la matriz de costos y la gestión de riesgos, ahora restringiremos el sistema LEXMED a la decisión de dos valores entre el diagnóstico de apendicitis y NSAP. Como posibles tratamientos, solo operación con probabilidad p1 y ambulatoria se puede elegir una observación con probabilidad p2 . La matriz de costos es por lo tanto un 2×2 matriz de la forma

$$\begin{matrix} & 0 & k2 \\ k1 & 0 & \end{matrix}$$

Los dos ceros en la diagonal representan la operación de decisión correcta en el caso de apendicitis y observación ambulatoria para NSAP. El parámetro k2 representa los costos esperados que ocurren cuando un paciente sin un apéndice inflamado es Operado en. Este error se denomina falso positivo. Por otra parte, la decisión

la observación ambulatoria en caso de apendicitis es un falso negativo. El vector de probabilidad $(p_1, p_2)^T$ ahora se multiplica por esta matriz y se obtiene el vector

$$(k_2 p_2, k_1 p_1)^T$$

con el coste adicional medio de los dos tratamientos posibles. Debido a que la decisión solo tiene en cuenta la relación de los dos componentes, el vector se puede multiplicar por cualquier factor escalar. Elegimos $1/k_1$ y obtenemos $((k_2/k_1)p_2, p_1)$.

Por tanto, aquí sólo es relevante la relación $k = k_2/k_1$. El mismo resultado se obtiene con la matriz de costos más simple

$$\begin{pmatrix} 0 & k & 1 \\ 0 & & \end{pmatrix},$$

que solo contiene la variable k . Este parámetro es muy importante porque determina la gestión del riesgo. Cambiando k podemos ajustar el "punto de trabajo" del sistema de diagnosis. Para $k \rightarrow \infty$ el sistema se pone en un escenario de riesgo extremo porque nunca se operará a ningún paciente, con la consecuencia de que no da clasificaciones de falsos positivos, pero sí muchos falsos negativos. En el caso de $k = 0$, las condiciones son exactamente inversas y se opera a todos los pacientes.

7.3.6 Rendimiento

LEXMED está diseñado para su uso en una práctica médica o ambulancia. Los requisitos previos para el uso de LEXMED son dolor abdominal agudo durante varias horas (pero menos de cinco días). Además, LEXMED está (actualmente) especializado en apendicitis, lo que significa que para otras enfermedades el sistema contiene muy poca información.

En el marco de un estudio prospectivo, se creó una base de datos representativa con 185 casos en el Hospital 14 Nothelfer. Contiene los pacientes del hospital que acudieron a la clínica después de varias horas de dolor abdominal agudo y sospecha de apendicitis. De estos pacientes se anotan los síntomas y el diagnóstico (comprobado a partir de una muestra de tejido en el caso de una operación).

Si los pacientes eran dados de alta para irse a casa (sin operación) después de una estadía de varias horas o de 1 a 2 días con poca o ninguna queja, luego se preguntaba por teléfono si el paciente seguía sin síntomas o si se encontró un diagnóstico positivo en tratamiento posterior.

Para simplificar la representación y hacer una mejor comparación con estudios similares, LEXMED se restringió a la distinción de dos valores entre apendicitis y NSAP, como se describe en la Secc. 7.3.5. Ahora k se varía entre cero e infinito y para cada valor de k se miden la sensibilidad y la especificidad frente a los datos de prueba.

Medidas de sensibilidad

$$P(\text{clasificado como positivo} | \text{positivo}) = \frac{|\text{positivo y clasificado positivo}|}{|\text{positivo}|},$$

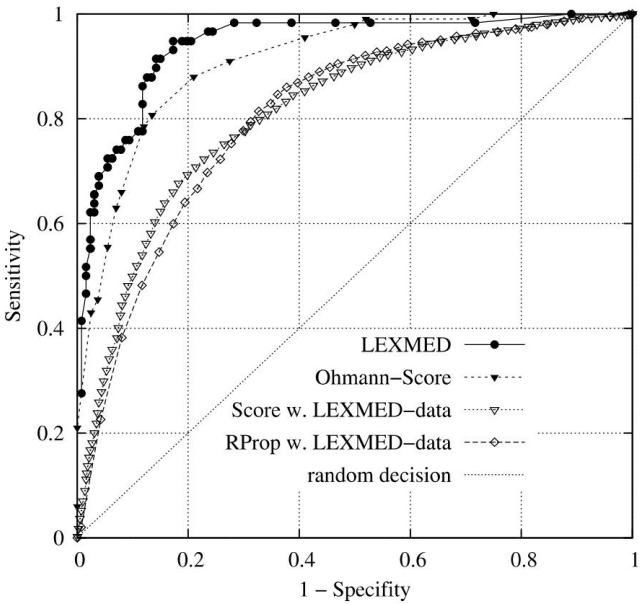


Fig. 7.10 Curva ROC de LEXMED comparada con la puntuación de Ohmann y dos modelos adicionales

es decir, la porción relativa de casos positivos que están correctamente identificados. Indica la sensibilidad del sistema de diagnóstico. La especificidad, por otro lado, mide

$$P(\text{clasificado negativo}|\text{negativo}) = \frac{|\text{negativo y clasificado negativo}|}{|\text{negativo}|},$$

es decir, la porción relativa de casos negativos que están correctamente identificados.

Damos los resultados de la sensibilidad y la especificidad en la figura 7.10 para $0 \leq k < \infty$. Esta curva se denomina curva ROC o característica operativa del receptor. Antes de pasar al análisis de la calidad de LEXMED, unas palabras sobre el significado de la curva ROC. La línea que divide el diagrama en diagonal se dibuja para orientarse. Todos los puntos de esta línea corresponden a una decisión aleatoria. Por ejemplo, el punto (0,2, 0,2) corresponde a una especificidad de 0,8 con una sensibilidad de 0,2. Podemos llegar a esto con bastante facilidad clasificando un nuevo caso, sin mirarlo, con probabilidades de 0,2 para positivo y 0,8 para negativo. Por lo tanto, todo sistema de diagnóstico basado en el conocimiento debe generar una ROC que se encuentre claramente por encima de la diagonal.

Los valores extremos de la curva ROC también son interesantes. En el punto (0, 0) las tres curvas se cruzan. El sistema de diagnóstico correspondiente clasificaría todos los casos como negativos. El otro valor extremo (1, 1) corresponde a un sistema que decidiría hacer la operación para cada paciente y por lo tanto tiene una sensibilidad de 1. Podríamos llamar a la curva ROC la curva característica de los sistemas de diagnóstico de dos valores. El sistema de diagnóstico ideal tendría una curva característica que consistiera únicamente en el punto (0, 1) y, por lo tanto, tendría una especificidad del 100 % y una sensibilidad del 100 %.

Ahora analicemos la curva ROC. Con una sensibilidad del 88 %, LEXMED alcanza una especificidad del 87 % ($k = 0,6$). A modo de comparación, se proporciona la puntuación de Ohmann, una puntuación establecida y bien conocida para la apendicitis [OMYL96, ZSR+99]. Debido a que LEXMED está por encima o a la izquierda de la puntuación de Ohmann en casi todas partes, su calidad promedio de diagnóstico es claramente mejor. Esto no es sorprendente porque los puntajes son simplemente demasiado débiles para modelar proposiciones interesantes. Insecto. 8.6 y en el ejercicio 8.16 de la página 219 mostraremos que las puntuaciones son equivalentes al caso especial de bayesiano ingenuo, es decir, a la suposición de que todos los síntomas son independientes por pares cuando se conoce el diagnóstico. Sin embargo, al comparar LEXMED con puntajes, debe mencionarse que se usó una base de datos estadísticamente representativa para el puntaje de Ohmann, pero para LEXMED se usó una base de datos no representativa mejorada con conocimiento. Para tener una idea de la calidad de los datos LEXMED en comparación con los datos de Ohmann, se calculó una puntuación lineal utilizando el método de mínimos cuadrados (consulte la Sección 9.4.1), que también se dibuja para comparar. Además, se entrenó una red neuronal sobre los datos LEXMED con el algoritmo RProp (ver Secc. 9.5). La fuerza de combinar datos y conocimiento experto se muestra claramente en la diferencia entre la curva LEXMED y las curvas del sistema de puntuación y el algoritmo RProp.

7.3.7 Áreas de aplicación y experiencias

LEXMED no debe reemplazar el juicio de un cirujano experimentado. Sin embargo, debido a que no siempre hay un especialista disponible en un entorno clínico, una consulta LEXMED ofrece una segunda opinión sustantiva. Especialmente interesante y valiosa es la aplicación del sistema en una ambulancia clínica y para médicos generales.

La capacidad de aprendizaje de LEXMED, que permite tener en cuenta más síntomas, más datos de pacientes y más reglas, también presenta nuevas posibilidades en la clínica. Para grupos especialmente raros que son difíciles de diagnosticar, por ejemplo, niños menores de seis años, LEXMED puede usar datos de pediatras u otras bases de datos especiales, para ayudar incluso a cirujanos experimentados.

Además del uso directo en el diagnóstico, LEXMED también admite medidas de garantía de calidad. Por ejemplo, las compañías de seguros pueden comparar la calidad del diagnóstico de los hospitales con la de los sistemas expertos. Al seguir desarrollando la matriz de costos creada en LEXMED (con el consentimiento de los médicos, el seguro y los pacientes), será más fácil comparar la calidad de los diagnósticos médicos, los diagnósticos computarizados y otras instituciones médicas.

LEXMED ha apuntado a una nueva forma de construir sistemas automáticos de diagnóstico. Utilizando el lenguaje de la teoría de la probabilidad y el algoritmo MaxEnt, el conocimiento derivado de forma inductiva y estadística se combina con el conocimiento de los expertos y de la literatura. El enfoque basado en modelos probabilísticos es teóricamente elegante, de aplicación general y ha dado muy buenos resultados en un pequeño estudio.

LEXMED ha estado en uso práctico en el Hospital 14 Nothelfer en Weingarten desde 1999 y ha funcionado muy bien allí. También está disponible en www.lexmed.de, sin garantía, por supuesto. Su calidad de diagnóstico es comparable con la de un cirujano experimentado y, por lo tanto, es mejor que la de un médico general promedio o la de un médico sin experiencia en la clínica.

A pesar de este éxito, se ha hecho evidente que es muy difícil comercializar un sistema de este tipo en el sistema médico alemán. Una de las razones de esto es que no existe un mercado libre para promover una mejor calidad (aquí mejores diagnósticos) a través de sus mecanismos de selección. Además, en medicina aún no ha llegado el momento de un uso generalizado de técnicas inteligentes, ni siquiera en 2010. Una causa de esto podría ser las enseñanzas conservadoras a este respecto en las facultades de las escuelas de medicina alemanas.

Otro problema es el deseo de muchos pacientes de recibir asesoramiento y atención personal por parte del médico, junto con el temor de que, con la introducción de sistemas expertos, el paciente solo se comuniquen con la máquina. Este temor, sin embargo, es totalmente infundado. Incluso a largo plazo, los sistemas médicos expertos no pueden reemplazar al médico.

Sin embargo, al igual que la cirugía láser y la resonancia magnética, se pueden utilizar ventajosamente para todos los participantes. Desde el primer sistema de diagnóstico informático médico de Dombal en 1972, han pasado casi 40 años. Queda por ver si la medicina esperará otros 40 años hasta que el diagnóstico por computadora se convierta en una herramienta médica establecida.

7.4 Razonamiento con Redes Bayesianas

Un problema con el razonamiento usando la probabilidad en la práctica ya fue señalado en la Secc. 7.1. Si se utilizan d variables $X_1, \dots, X_d$ con n valores cada una, entonces los d valores totales asociados. Esto la distribución de probabilidad tiene un significado que, en el peor de los casos, el uso de memoria n y el tiempo de cálculo para determinar las probabilidades especificadas crece exponencialmente con el número de variables.

En la práctica, las aplicaciones suelen estar muy estructuradas y la distribución contiene muchas redundancias. Esto significa que puede reducirse considerablemente con los métodos apropiados. El uso de redes bayesianas ha demostrado su poder aquí y es una de las técnicas de IA que se han utilizado con éxito en la práctica. Las redes bayesianas utilizan el conocimiento sobre la independencia de las variables para simplificar el modelo.

7.4.1 Variables independientes

En el caso más simple, todas las variables son independientes por pares y se da el caso de que

$$P(X_1, \dots, X_d) = P(X_1) \cdot P(X_2) \cdots P(X_d).$$

Todas las entradas en la distribución se pueden calcular a partir de los valores d $P(X_1), \dots, P(X_d)$. Sin embargo, las aplicaciones interesantes por lo general no se pueden modelar porque las probabilidades condicionales se vuelven triviales.¹³ Debido a

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A) P(B)}{P(B)} = P(A)$$

¹³En el método ingenuo de Bayes, se supone la independencia de todos los atributos, y este método se ha aplicado con éxito a la clasificación de textos (ver Secc. 8.6).

todas las probabilidades condicionales se reducen a las probabilidades a priori. La situación se vuelve más interesante cuando solo una parte de las variables son independientes o dependientes bajo ciertas condiciones. Para el razonamiento en IA, las dependencias entre variables son importantes y deben utilizarse.

Nos gustaría delinear el razonamiento con redes bayesianas a través de un ejemplo simple y muy ilustrativo de J. Pearl [Pea88], que se hizo conocido a través de [RN10] y ahora es un conocimiento básico de IA.

Ejemplo 7.7 (Alarma-Ejemplo) Bob, que es soltero, ha instalado un sistema de alarma en su casa para protegerse contra los ladrones. Bob no puede oír la alarma cuando está trabajando en la oficina. Por lo tanto, ha pedido a sus dos vecinos, John en la casa de al lado a la izquierda y Mary en la casa de la derecha, que lo llamen a su oficina si escuchan su alarma. Después de algunos años, Bob sabe qué tan confiables son John y Mary y modela su comportamiento de llamadas utilizando la probabilidad condicional de la siguiente manera.¹⁴

$$\begin{aligned} P(J|A) &= 0,90 \quad P(M|A) = 0,70, \quad P(J|\neg A) = 0,05 \\ P(M|\neg A) &= 0,01. \end{aligned}$$

Debido a que Mary tiene problemas de audición, no escucha la alarma con más frecuencia que John. Sin embargo, John a veces confunde la alarma de la casa de Bob con las alarmas de otras casas. La alarma se dispara por un robo, pero también puede ser disparada por un terremoto (débil), lo que puede provocar una falsa alarma porque Bob solo quiere saber sobre robos mientras está en su oficina. Estas relaciones están modeladas por

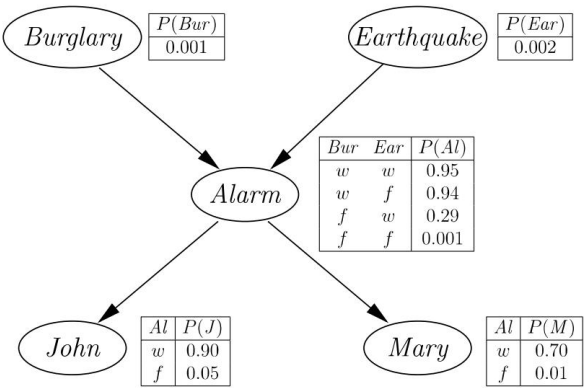
$$\begin{aligned} P(A|Rebaba, Oído) &= 0,95, \\ P(A|Bur, \neg Oído) &= 0,94, \\ P(A|\neg Bur, Ear) &= 0,29, \\ P(A|\neg Bur, \neg Oído) &= 0,001, \end{aligned}$$

así como las probabilidades a priori $P(Bur) = 0.001$ y $P(Ear) = 0.002$. Estas dos variables son independientes porque los terremotos no hacen planes basados en los hábitos de los ladrones y, por el contrario, no hay forma de predecir los terremotos, por lo que los ladrones no tienen la oportunidad de establecer su horario en consecuencia.

Las consultas ahora se realizan contra esta base de conocimiento. Por ejemplo, Bob podría estar interesado en $P(Bur|J \quad M)$, $P(J|Bur)$ o $P(M|Bur)$. Es decir, quiere saber con qué sensibilidad reaccionan las variables J y M ante un informe de robo.

¹⁴Las variables binarias J y M representan los dos eventos "John llama" y "Mary llama", respectivamente, A para "sonidos de sirena de alarma", Bur para "robo" y Ear para "terremoto".

Fig. 7.11 Red bayesiana para el ejemplo de alarma con los CPT asociados



7.4.2 Representación gráfica del conocimiento como una red bayesiana

Podemos simplificar mucho el trabajo práctico representando gráficamente el conocimiento que se formula como probabilidad condicional. La figura 7.11 muestra la red bayesiana para el ejemplo de alarma. Cada nodo en la red representa una variable y cada borde dirigido una declaración de probabilidad condicional. La arista de Al a J, por ejemplo, representa los dos valores $P(J|Al)$ y $P(J|\neg Al)$, que se dan en forma de tabla, la llamada CPT (tabla de probabilidad condicional). El CPT de un nodo enumera todas las probabilidades condicionales de la variable del nodo condicionada en todos los nodos conectados por aristas entrantes.

Mientras estudiamos la red, podríamos preguntarnos por qué no se incluyen otros bordes además de los cuatro que se dibujan. Los dos nodos Bur y Ear no están vinculados ya que las variables son independientes. Todos los demás nodos tienen un nodo principal, lo que hace que el razonamiento sea un poco más complejo. Primero necesitamos el concepto de independencia condicional.

7.4.3 Independencia condicional

Análogamente a la independencia de las variables aleatorias, damos

Definición 7.6 Dos variables A y B se llaman condicionalmente independientes, dada C si

$$P(A, B|C) = P(A|C) \cdot P(B|C).$$

Esta ecuación es cierta para todas las combinaciones de valores de las tres variables (es decir, para la distribución), que vemos en la notación. Ahora observamos los nodos J y M en el ejemplo de alarma, que tienen el nodo principal común Al. Si Juan y María

reaccionan independientemente a una alarma, entonces las dos variables J y M son independientes dado Al, es decir:

$$P(J, M | Al) = P(J | Al) \cdot P(M | Al).$$

Si se conoce el valor de Al, por ejemplo porque se disparó una alarma, entonces las variables J y M son independientes (bajo la condición $Al = w$). Debido a la independencia condicional de las dos variables J y M, no se agrega ningún borde entre estos dos nodos. Sin embargo, J y M no son independientes (vea el Ejercicio 7.11 en la página 159).

Bastante similar es la relación entre las dos variables J y Bur, porque Juan no reacciona ante un robo, sino ante la alarma. Esto podría deberse, por ejemplo, a que un muro alto bloquea su vista de la propiedad de Bob, pero aún puede escuchar la alarma. Así J y Bur son independientes dados Al y

$$P(J, Bur | Al) = P(J | Al) \cdot P(Bur | Al).$$

Ante una alarma, las variables J y Oído, M y Bur, así como M y Oído también son independientes. Para calcular con independencia condicional, son útiles las siguientes caracterizaciones, que son equivalentes a la definición anterior:

Teorema 7.5 Las siguientes ecuaciones son equivalentes por pares, lo que significa que cada ecuación individual describe la independencia condicional de las variables A y B dada C.

$$P(A, B | C) = P(A | C) \cdot P(B | C), P(A | B, C) = \quad (7.16)$$

$$P(A | C), P(B | A, C) = P(B | C). \quad (7.17)$$

$$(7.18)$$

Demostración Por un lado, usando la independencia condicional (7.16) podemos concluir que

$$P(A, B, C) = P(A, B | C)P(C) = P(A | C)P(B | C)P(C).$$

Por otro lado, la regla del producto nos da

$$P(A, B, C) = P(A | B, C)P(B | C)P(C).$$

Así $P(A | B, C) = P(A | C)$ es equivalente a (7.16). Obtenemos (7.18) de forma análoga intercambiando A y B en esta derivación.

7.4.4 Aplicación práctica

Ahora volvamos al ejemplo de la alarma y mostramos cómo se puede usar la red bayesiana de la figura 7.11 en la página 146 para razonar. Bob está interesado, por ejemplo, en la sensibilidad de sus dos reporteros de alarma, John y Mary, es decir, en $P(J | \text{Bur})$ y $P(M | \text{Bur})$. Sin embargo, los valores $P(\text{Bur} | J)$ y $P(\text{Bur} | M)$, así como $P(\text{Bur} | J, M)$ son aún más importantes para él. Empezamos con $P(J | \text{Bur})$ y calculamos

$$P(J | \text{Fresa}) = \frac{P(J, \text{Bur})}{P(\text{fresa})} = \frac{P(J, \text{Bur}, \text{Al}) + P(J, \text{Bur}, \neg \text{Al})}{P(\text{fresa})} \quad (7.19)$$

y

$$P(J, \text{Bur}, \text{Al}) = P(J | \text{Bur}, \text{Al})P(\text{Al} | \text{Bur})P(\text{Bur}) = P(J | \text{Al})P(\text{Al} | \text{Bur})P(\text{Bur}), \quad (7.20)$$

donde para las dos últimas ecuaciones hemos utilizado la regla del producto y la independencia condicional de J y Bur dada Al . Insertado en (7.19) obtenemos

$$\begin{aligned} P(J | \text{Fresa}) &= \frac{P(J | \text{Al})P(\text{Al} | \text{Bur})P(\text{Bur}) + P(J | \neg \text{Al})P(\neg \text{Al} | \text{Bur})P(\text{Bur})}{P(\text{fresa})} \\ &= P(J | \text{Al})P(\text{Al} | \text{Bur}) + P(J | \neg \text{Al})P(\neg \text{Al} | \text{Bur}). \end{aligned} \quad (7.21)$$

Aquí faltan $P(\text{Al} | \text{Bur})$ y $P(\neg \text{Al} | \text{Bur})$. Por lo tanto calculamos

$$\begin{aligned} P(\text{Al} | \text{Bur}) &= \frac{P(\text{Al}, \text{Bur})}{P(\text{fresa})} = \frac{P(\text{Al}, \text{Fresa}, \text{Oreja}) + P(\text{Al}, \text{Fresa}, \neg \text{Oreja})}{P(\text{fresa})} \\ &= \frac{P(\text{Al} | \text{Bur}, \text{Oído})P(\text{Bur})P(\text{Oído}) + P(\text{Al} | \text{Bur}, \neg \text{Oído})P(\text{Bur})P(\neg \text{Oído})}{P(\text{fresa})} \\ &= P(\text{Al} | \text{Fresa}, \text{Oído})P(\text{Oído}) + P(\text{Al} | \text{Fresa}, \neg \text{Oído})P(\neg \text{Oído}) \\ &= 0,95 \cdot 0,002 + 0,94 \cdot 0,998 = 0,94 \end{aligned}$$

así como $P(\neg \text{Al} | \text{Bur}) = 0.06$ e inserte esto en (7.21) que da el resultado

$$P(J | \text{Bur}) = 0,9 \cdot 0,94 + 0,05 \cdot 0,06 = 0,849.$$

Análogamente calculamos $P(M | \text{Bur}) = 0,659$. Ahora sabemos que John llama alrededor del 85% de todos los robos y Mary alrededor del 66% de todos los robos. La probabilidad de que ambos llamen se calcula, por independencia condicional, como

$$P(J, M | \text{Bur}) = P(J | \text{Bur})P(M | \text{Bur}) = 0,849 \cdot 0,659 = 0,559.$$

Más interesante, sin embargo, es la probabilidad de que una llamada de John o Mary

$$\begin{aligned} P(J \mid M|Bur) &= P(\neg(\neg J, \neg M)|Bur) = 1 - P(\neg J, \neg M|Bur) \\ &= 1 - P(\neg J|Bur)P(\neg M|Bur) = 1 - 0,051 = 0,948. \end{aligned}$$

Bob recibe notificación de alrededor del 95% de todos los robos. Ahora para calcular $P(Bur|J)$, aplicamos la fórmula de Bayes, que nos da

$$P(Bur|J) = \frac{P(J|Fresa)P(Fresa)}{P(J)} = \frac{0,849 \cdot 0,001}{0,052} = 0,016.$$

Evidentemente, sólo alrededor del 1,6% de todas las llamadas de John se deben realmente a un robo. Debido a que la probabilidad de falsas alarmas es cinco veces menor para Mary, con $P(Bur|M) = 0,056$ tenemos una confianza significativamente mayor ante una llamada de Mary. Bob solo debería estar seriamente preocupado por su casa si ambos llaman, porque $P(Bur|J,M) = 0,284$ (vea el Ejercicio 7.11 en la página 159).

En (7.21) en la página 148 mostramos con

$$P(J|Bur) = P(J|Al)P(Al|Bur) + P(J|\neg Al)P(\neg Al|Bur)$$

cómo podemos "deslizar" una nueva variable. Esta relación se cumple en general para dos variables A y B dada la introducción de una variable adicional C y se denomina condicionamiento:

$$PAG(A|B) = \sum_c P(A|B, C=c)P(C=c|B).$$

Si además A y B son condicionalmente independientes dado C, esta fórmula se simplifica a

$$PAG(A|B) = \sum_c P(A|C=c)P(C=c|B).$$

7.4.5 Software para Redes Bayesianas

Daremos una breve introducción a dos herramientas utilizando el ejemplo de alarma. Ya estamos familiarizados con el sistema PIT. Ingresamos los valores de los CPT en sintaxis PIT en la ventana de entrada en línea en www.pit-systems.de. Después de la entrada que se muestra en la Fig. 7.12 en la página 150, recibimos la respuesta:

$$P([Einbruch=t] \mid [Juan=t] \text{ Y } [María=t]) = 0,2841.$$

Si bien PIT no es una herramienta de red bayesiana clásica, puede tomar probabilidades y consultas condicionales arbitrarias como entrada y calcular los resultados correctos. Se puede demostrar [Sch96] que en la entrada de CPT o reglas equivalentes, el principio MaxEnt implica las mismas independencias condicionales y, por lo tanto, también las mismas respuestas que una red bayesiana. Las redes bayesianas son, por lo tanto, un caso especial de MaxEnt.

```
1 var Alarma{t,f}, Robo{t,f}, Terremoto{t,f}, Juan{t,f}, Maria{t,f};
2
3 P([Terremoto=t]) = 0.002; 4 P([Robo=t]) =
0,001; 5 P([Alarma=t] | [Robo=t] Y
[Terremoto=t]) = 0,95; 6 P([Alarma=t] | [Robo=t] Y [Terremoto=f]) = 0,94; 7 P([Alarma=t] |
[Robo=f] Y [Terremoto=t]) = 0,29; 8 P([Alarma=t] | [Robo=f] Y [Terremoto=f]) = 0,001; 9
P([Juan=t] | [Alarma=t]) = 0,90; 10 P([Juan=t] | [Alarma=f]) = 0,05; 11 P([Maria=t] |
[Alarma=t]) = 0,70; 12 P([Maria=t] | [Alarma=f]) = 0,01; 13 14 QP([Robo=t] | [Juan=t] Y
[Maria=t]);
```

Fig. 7.12 Entrada PIT para el ejemplo de alarma

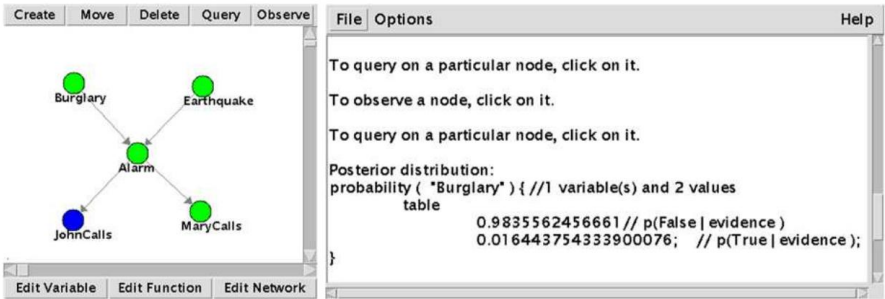


Fig. 7.13 La interfaz de usuario de JavaBayes: a la izquierda el editor gráfico y a la derecha la consola donde se dan las respuestas como salida

A continuación veremos JavaBayes [Coz98], un sistema clásico también disponible gratuitamente en Internet con la interfaz gráfica que se muestra en la figura 7.13. Con el editor de red gráfica, los nodos y los bordes se pueden manipular y los valores en los CPT se pueden editar. Además, los valores de las variables se pueden asignar con "Observe" y los valores de otras variables se pueden llamar con "Query". Las respuestas a las consultas aparecen en la ventana de la consola.

El sistema comercial profesional Hugin es significativamente más potente y conveniente. Por ejemplo, Hugin puede usar variables continuas además de variables discretas. También puede aprender redes bayesianas, es decir, generar la red de forma totalmente automática a partir de datos estadísticos (ver Secc. 8.5).

7.4.6 Desarrollo de Redes Bayesianas

Una red bayesiana compacta es muy clara y significativamente más informativa para el lector que una distribución de probabilidad completa. Además, requiere mucho menos

memoria. Para las variables $v_1, \dots, v_n$ con $|v_1|, \dots, |v_n|$ diferentes valores cada uno, la distribución tiene un total de

$$\prod_{i=1}^n (|v_i| - 1) + 1$$

entradas independientes. En el ejemplo de alarma, las variables son todas binarias. Así, para todas las variables $|v_i| = 2$, y la distribución tiene $2^5 - 1 = 31$ entradas independientes. Para calcular el número de entradas independientes para la red bayesiana, se debe determinar el número total de todas las entradas de todos los CPT. Para un nodo v_i con nodos padres k_i $e_{i1}, \dots, e_{ik_i}$, el CPT asociado tiene

$$\prod_{j=1}^{k_i} (|v_i| - 1) + 1$$

entradas. Entonces todos los CPT en la red juntos tienen

$$\sum_{i=1}^n \prod_{j=1}^{k_i} (|v_i| - 1) + 1 \quad (7.22)$$

entradas.15 Para el ejemplo de alarma, el resultado es entonces

$$2 + 2 + 4 + 1 + 1 = 10$$

entradas independientes que describen de forma única la red. La comparación en la complejidad de la memoria entre la distribución completa y la red bayesiana se vuelve más clara cuando asumimos que todas las n variables tienen el mismo número b de valores y cada nodo tiene k nodos principales.

Entonces (7.22) se puede simplificar y todos los CPT juntos tienen $n(b - 1)^k + 1$ entradas. La distribución completa contiene b^n , la ganancia no se puede obtener solo si el número $(b - 1)^k - 1$ entradas. un significado promedio de nodos principales es mucho menor que el número de variables. Esto significa que los nodos solo están conectados localmente.

Debido a la conexión local, la red se modulariza, lo que, como en la ingeniería de software, conduce a una reducción de la complejidad. En el ejemplo de alarma, el nodo de alarma separa los nodos Bur y Ear de los nodos J y M. También podemos ver esto claramente en el ejemplo de LEXMED .

15Para el caso de un nodo sin ancestros el producto en esta suma está vacío. Para esto sustituimos el valor 1 porque el CPT para nodos sin ancestros contiene, con su probabilidad a priori, exactamente un valor.

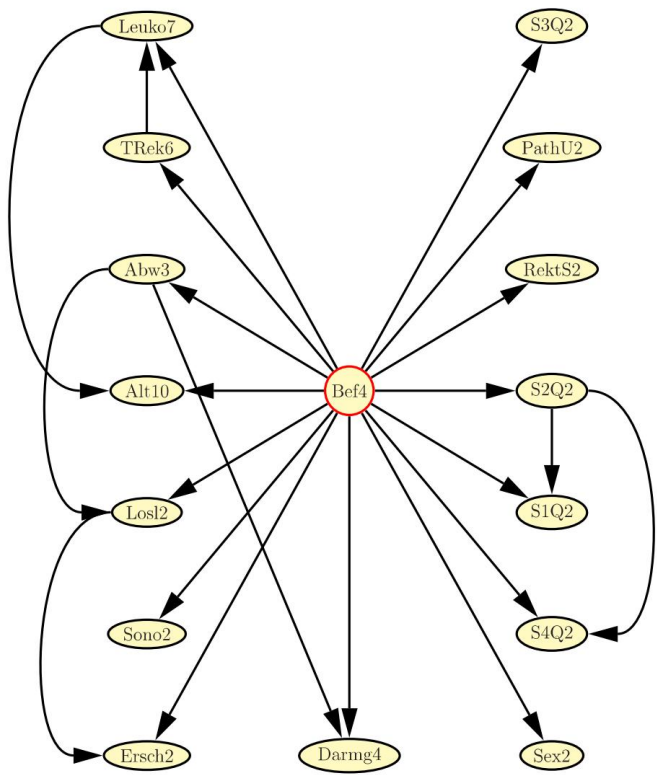


Fig. 7.14 Red bayesiana para la aplicación LEXMED

LEXMED como Red Bayesiana El sistema LEXMED descrito en la Secc. 7.3 también se puede modelar como una red bayesiana. Al hacer que las líneas exteriores finamente dibujadas estén dirigidas (dándoles flechas), el gráfico de independencia de la figura 7.8 en la página 137 se puede interpretar como una red bayesiana. La red resultante se muestra en la figura 7.14.

Insecto. 7.3.2 el tamaño de la distribución para LEXMED se calculó como el valor 20 643 839. La red bayesiana, por otro lado, se puede describir completamente con solo 521 valores. Este valor se puede determinar ingresando las variables de la Fig. 7.14 en (7.22) en la página 151. En el orden (Leuko, TRek, Gua, Age, Reb, Sono, Tapp, BowS, Sex, P4Q, P1Q, P2Q, RecP , Urin, P3Q, Diag4) calculamos

$$\begin{aligned} & \sum_{i=1}^n \sum_{j=1}^n (|v_i| - 1) \cdot |e_{ij}| = 6 \cdot 6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 7 \cdot 4 + 1 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 2 \cdot 4 = 1 \\ & + 3 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \\ & + 1 \cdot 4 + 1 = 521. \end{aligned}$$

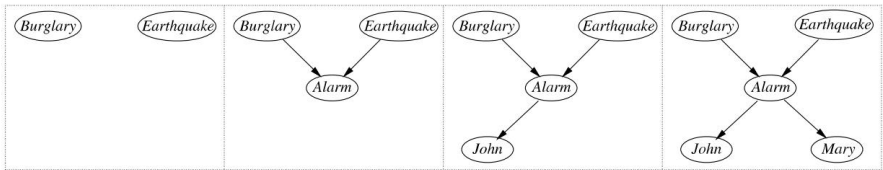


Fig. 7.15 Construcción por pasos de la red de alarma considerando la causalidad

Este ejemplo demuestra que es prácticamente imposible construir una distribución completa para aplicaciones reales. Una red bayesiana con 22 aristas y 521 valores de probabilidad, por otro lado, sigue siendo manejable.

Causalidad y estructura de la red

La construcción de una red bayesiana generalmente se realiza en dos etapas.

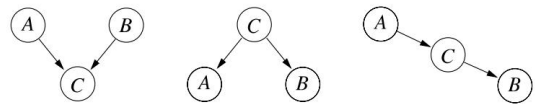
1. Diseño de la estructura de la red: Este paso generalmente se realiza manualmente y se describirá a continuación.
2. Introducir las probabilidades en los CPTs: Introducir manualmente los valores en el caso de muchas variables es muy tedioso. Si (como por ejemplo con LEXMED) hay una base de datos disponible, este paso se puede automatizar mediante la estimación de las entradas de CPT mediante el conteo de frecuencias.

Ahora describiremos la construcción de la red en el ejemplo de alarma (ver Fig. 7.15). Al principio conocemos las dos causas Robo y Terremoto y los dos síntomas Juan y María. Sin embargo, debido a que John y Mary no reaccionan directamente ante un ladrón o un terremoto, sino solo ante la alarma, es apropiado agregar esto como una variable adicional que Bob no puede observar. El proceso de agregar aristas comienza con las causas, es decir, con las variables que no tienen nodos principales. Primero elegimos Robo y después Terremoto. Ahora debemos comprobar si Terremoto es independiente de Robo. Esto se da y, por lo tanto, no se agrega ventaja de Robo a Terremoto. Debido a que la Alarma depende directamente de Robo y Terremoto, estas variables se eligen a continuación y se agrega un borde de Robo y Terremoto a Alarma. Entonces elegimos a John. Debido a que Alarma y John no son independientes, se agrega un borde de alarma a John. Lo mismo es cierto para María. Ahora debemos verificar si John es condicionalmente independiente de Robo dado Alarma. Si este no es el caso, entonces se debe insertar otro borde desde Robo hasta Juan. También debemos comprobar si se necesitan aristas de Terremoto a Juan y de Robo o Terremoto a María. Debido a la independencia condicional, estas cuatro aristas no son necesarias. Los bordes entre John y Mary también son innecesarios porque John y Mary son condicionalmente independientes dada la Alarma.

La estructura de la red bayesiana depende en gran medida del orden de las variables elegidas. Si el orden de las variables se elige para reflejar la relación causal comenzando con las causas y continuando con las variables de diagnóstico, entonces el resultado será una red simple. De lo contrario, la red puede contener muchos más bordes.

Estas redes no causales suelen ser muy difíciles de entender y tienen un mayor

Fig. 7.16 No hay arista entre A y B si son independientes (izquierda) o condicionalmente independientes (centro, derecha)



complejidad para el razonamiento. El lector puede consultar el Ejercicio 7.11 en la página 159 para una mejor comprensión.

7.4.7 Semántica de Redes Bayesianas

Como vimos en la sección anterior, no se agrega borde a una red bayesiana entre dos variables A y B cuando A y B son independientes o condicionalmente independientes dada una tercera variable C. Esta situación se representa en la figura 7.16 .

Ahora requerimos que la red bayesiana no tenga ciclos y asumimos que las variables están numeradas de tal manera que ninguna variable tiene un número más bajo que cualquier variable que la preceda. Esto siempre es posible cuando la red no tiene ciclos.¹⁶ Entonces, al usar todas las independencias condicionales, tenemos

$$P(X_n|X_1,...,X_{n-1}) = P(X_n|Padres(X_n)).$$

Por lo tanto, esta ecuación es una proposición de que una variable arbitraria X_i en una red bayesiana es condicionalmente independiente de sus ancestros, dados sus padres. La proposición algo más general representada en la figura 7.17 en la página 155 se puede enunciar de manera compacta

como

Teorema 7.6 Un nodo en una red bayesiana es condicionalmente independiente de todos los nodos no sucesores, dados sus padres.

Ahora podemos simplificar en gran medida la regla de la cadena ((7.1) en la página 120):

$$P(X_1,...,X_n) =$$

$$\prod_{i=1}^n P(X_i | X_1 \dots, X_{i-1}) =$$

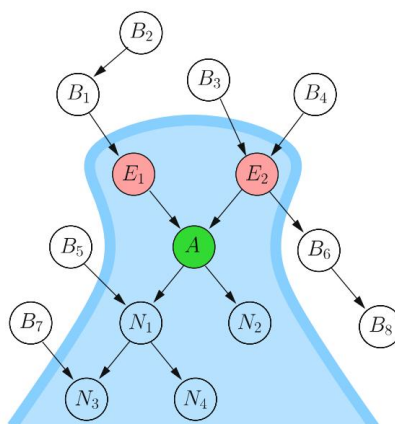
$$\prod_{i=1}^n P(X_i | Padres(X_i)). \quad (7.23)$$

Usando esta regla podríamos, por ejemplo, escribir (7.20) en la página 148 directamente

$$P(J,Bur,AI) = P(J | AI)P(AI|Bur)P(Bur).$$

¹⁶Si, por ejemplo, tres nodos X_1, X_2, X_3 forman un ciclo, entonces están los bordes $(X_1, X_2), (X_2, X_3)$ y (X_3, X_1) donde X_1 tiene a X_3 como sucesor.

Fig. 7.17 Ejemplo de independencia condicional en una red bayesiana. Si se dan los nodos principales E_1 y E_2 , entonces todos los nodos no sucesores $B_1, \dots, B_8$ son independientes de A .



Ahora conocemos los conceptos y fundamentos más importantes de la red bayesiana. obras. Resumámoslos [Jen01]:

Definición 7.7 Una red bayesiana se define por: • Un conjunto de variables y un conjunto de aristas dirigidas entre estas variables. • Cada variable tiene un número finito de valores posibles.

- Las variables junto con las aristas forman un gráfico acíclico dirigido (DAG).

Un DAG es un grafo sin ciclos, es decir, sin caminos de la forma $(A, \dots, A)$.

- Para cada variable A el CPT (es decir, la tabla de probabilidades condicionales) se da $P(A|\text{Parents}(A))$.

Dos variables A y B se llaman condicionalmente independientes dado C si $P(A, B|C) = P(A|C) \cdot P(B|C)$ o, de manera equivalente, si $P(A|B, C) = P(A|C)$.

Además de las reglas fundamentales de cálculo de probabilidades, las siguientes reglas también son ciertas:

Teorema de Bayes:
$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Marginación: $P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A)$

Condicionamiento: $P(A|B) = P(A|B, C = c)P(C = c|B)$

Una variable en una red bayesiana es condicionalmente independiente de todas las variables no sucesoras dadas sus variables principales. Si $X_1, \dots, X_{n-1}$ no son sucesores de X_n , tenemos $P(X_n|X_1, \dots, X_{n-1}) = P(X_n|\text{Parents}(X_n))$. Esta condición debe respetarse durante la construcción de una red.

Durante la construcción de una red bayesiana, las variables deben ordenarse según la causalidad. Primero las causas, luego las variables ocultas y por último las variables de diagnóstico.

Regla de la cadena: $P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | \text{Padres}(X_i))$

En [Pea88] y [Jen01] se introduce el término separación d para las redes bayesianas, de las que se sigue un teorema similar al Teorema 7.6 de la página 154. Nos abstendremos de introducir este término y así llegar a una representación algo más simple, aunque teóricamente no tan limpia.

7.5 Resumen

De una manera que refleja la tendencia sostenida y prolongada hacia los sistemas probabilísticos, hemos introducido la lógica probabilística para razonar con conocimiento incierto. Después de introducir el lenguaje (lógica proposicional aumentada con probabilidades o intervalos de probabilidad), elegimos el enfoque natural, aunque inusual, a través del método de máxima entropía como punto de entrada y mostramos cómo podemos modelar el razonamiento no monótono con este método. Luego se introdujeron las redes bayesianas como un caso especial del método MaxEnt.

¿Por qué las redes bayesianas son un caso especial de MaxEnt? Al construir una red bayesiana, se hacen suposiciones sobre la independencia que son innecesarias para el método MaxEnt. Además, al construir una red bayesiana, todos los CPT deben completarse por completo para que se pueda construir una distribución de probabilidad completa. De lo contrario, el razonamiento es restringido o imposible. Con MaxEnt, en cambio, el desarrollador puede formular todo el conocimiento que tiene a su disposición en forma de probabilidades. MaxEnt luego completa el modelo y genera la distribución. Incluso si (por ejemplo, al entrevistar a un experto) solo se dispone de proposiciones vagas, esto se puede modelar adecuadamente. Una proposición como "Estoy bastante seguro de que A es verdadero". puede, por ejemplo, modelarse usando $P(A) \in [0.6, 1]$ como un intervalo de probabilidad. Al construir una red bayesiana, se debe dar un valor concreto para la probabilidad, si es necesario adivinando. Esto significa, sin embargo, que el experto o el desarrollador introducen información ad hoc en el sistema. Otra ventaja de MaxEnt es la posibilidad de formular proposiciones (casi) arbitrarias. Para las redes bayesianas, se deben completar los CPT.

La libertad que tiene el desarrollador al modelar con MaxEnt puede ser una desventaja (especialmente para un principiante) porque, a diferencia del enfoque bayesiano, no está necesariamente claro qué conocimiento debe modelarse. Cuando se modela con redes bayesianas, el enfoque es bastante claro: de acuerdo con las dependencias causales, desde las causas hasta los efectos, se ingresa un borde tras otro en la red al probar la independencia condicional.¹⁷ Al final, todos los CPT se llenan con valores.

Sin embargo, son posibles las siguientes combinaciones interesantes de los dos métodos: comenzamos construyendo una red de acuerdo con la metodología bayesiana, ingresamos todos los bordes en consecuencia y luego llenamos los CPT con valores. Si ciertos valores para los CPT no están disponibles, entonces pueden ser reemplazados por intervalos o por otras fórmulas de lógica probabilística. Naturalmente, tal red, o mejor: un conjunto de reglas, no

¹⁷Esto tampoco es siempre tan simple.

ya no tiene la semántica especial de una red bayesiana. Luego debe ser procesado y completado por un sistema MaxEnt.

Sin embargo, la capacidad de usar MaxEnt con conjuntos de reglas arbitrarias tiene un inconveniente. De manera similar a la situación en la lógica, tales conjuntos de reglas pueden ser inconsistentes. Por ejemplo, las dos reglas $P(A) = 0,7$ y $P(A) = 0,8$ son incompatibles. Si bien el PIT del sistema MaxEnt, por ejemplo, puede reconocer la inconsistencia, no puede dar una pista sobre cómo eliminar el problema.

Presentamos el sistema médico experto LEXMED, una aplicación clásica para razonar con conocimiento incierto, y mostramos cómo se puede modelar e implementar utilizando MaxEnt y redes bayesianas, y cómo estas herramientas pueden reemplazar la puntuación lineal bien establecida, pero demasiado débil. sistemas utilizados en medicina.¹⁸

En el ejemplo de LEXMED mostramos que es posible construir un sistema experto para razonar bajo incertidumbre que sea capaz de descubrir (aprender) conocimiento de los datos en una base de datos. Daremos más información sobre los métodos de aprendizaje de las redes bayesianas en el Cap. 8, después de que se hayan sentado las bases necesarias para el aprendizaje automático.

Hoy en día, el razonamiento bayesiano es un campo grande e independiente, que solo podemos describir brevemente aquí. Hemos dejado completamente de lado el manejo de variables continuas. Para el caso de variables aleatorias normalmente distribuidas existen procedimientos y sistemas. Sin embargo, para distribuciones arbitrarias, la complejidad computacional es un gran problema. Además de las redes dirigidas que se basan en gran medida en la causalidad, también existen redes no dirigidas. Conectado con esto hay una discusión sobre el significado y la utilidad de la causalidad en las redes bayesianas. El lector interesado puede consultar excelentes libros de texto como [Pea88, Jen01, Whi96, DHS01], así como las actas de la conferencia anual de la Asociación para la Incertidumbre en Inteligencia Artificial (AUAI) (www.auai.org).

7.6 Ejercicios

Ejercicio 7.1 Demostrar la proposición del Teorema 7.1 de la página 117.

Ejercicio 7.2 El aficionado a la jardinería Max quiere analizar estadísticamente su cosecha anual de guisantes. Por cada vaina de guisante que recoge, mide su longitud x_i en centímetros y su peso y_i en gramos. Divide los guisantes en dos clases, los buenos y los malos (vainas vacías). Los datos medidos (x_i, y_i) son

	x 122334456 y 234455666	x 4667 y 2233
buenos guisantes:		guisantes malos:

¹⁸ En la secc. 8.6 y en el ejercicio 8.16 de la página 219 mostraremos que las puntuaciones son equivalentes al caso especial naive Bayes, es decir, a la suposición de que todos los síntomas son condicionalmente independientes dado el diagnóstico.

- (a) A partir de los datos, calcule las probabilidades $P(y > 3 | \text{Clase} = \text{bueno})$ y $P(y \leq 3 | \text{Clase} = \text{buena})$. Luego use la fórmula de Bayes para determinar $P(\text{Clase} = \text{bueno} | y > 3)$ y $P(\text{Clase} = \text{buena} | y \leq 3)$.
- (b) ¿Cuál de las probabilidades calculadas en el subproblema (a) contradice el enunciado ¿"Todos los buenos guisantes pesan más de 3 gramos"?

Ejercicio 7.3 Se supone que debes predecir el clima de la tarde usando unos simples valores meteorológicos desde la mañana de este día. El cálculo de probabilidad clásico para esto se requiere un modelo completo, el cual se da en la siguiente tabla.

Cielo	Bar	Prec	P (Cielo, Bar, Prec)	
Claro	Creciente Seco		0.40	Cielo: El cielo está despejado o nublado por la mañana
Claro	Creciente Lluvia	0.07		
Claro	Quedarse Seco	0.08		
Claro	Cayendo Lloviendo	0.10		Barra: Barómetro subiendo o cayendo en la mañana
Bewölkt	Rising Dry	Bewölkt	0.09	Prec: Lluvia o seco en el tarde
Rising	Raining	0.11		
Bewölkt	Falling Dry	0.03		

- (a) ¿Cuántos eventos hay en la distribución para estas tres variables?
- (b) Calcule $P(\text{Prec} = \text{seco} | \text{Cielo} = \text{despejado}, \text{Bar} = \text{ascendente})$.
- (c) Calcule $P(\text{Prec} = \text{lluvia} | \text{Cielo} = \text{nublado})$.
- (d) ¿Qué harías si faltara la última fila de la tabla?

Ejercicio 7.4 En un concurso televisivo, el concursante debe elegir entre tres puertas cerradas. Detrás de una puerta espera el premio: un coche. Detrás de los dos del otro las puertas son cabras. El concursante elige una puerta, por ejemplo, la número uno. El anfitrión, que sabe dónde está el coche, abre otra puerta, por ejemplo la número tres, y aparece una cabra. El concursante ahora tiene la oportunidad de elegir entre los dos restantes. puertas (una y dos). ¿Cuál es la mejor opción desde su punto de vista? Quedarse con la puerta que eligió originalmente o cambiar a la otra puerta cerrada?

Ejercicio 7.5 Utilizando el método del multiplicador de Lagrange, demuestre que, sin restricciones, la distribución uniforme $p_1 = p_2 = \dots = p_n = 1/n$ representa la máxima entropía. No olvide la restricción implícita siempre presente $p_1 + p_2 + \dots + p_n = 1$. ¿Cómo podemos mostrar este mismo resultado usando la indiferencia?

Ejercicio 7.6 Utilice el sistema PIT (www.pit-systems.de) para calcular MaxEnt solución para $P(B)$ bajo la restricción $P(A) = \alpha$ y $P(B|A) = \beta$. ¿Qué desventaja del PIT, en comparación con el cálculo manual, observa aquí?

Ejercicio 7.7 Dadas las restricciones $P(A) = \alpha$ y $P(B) = \beta$, calcule manualmente $P(B)$ usando el método MaxEnt. Use PIT para verificar su solución.

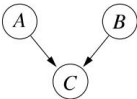
Ejercicio 7.8 Dadas las restricciones de (7.10), (7.11), (7.12): $p_1 + p_2 = \alpha$, $p_1 + p_3 = \gamma$, $p_1 + p_2 + p_3 + p_4 = 1$. Demuestre que $p_1 = \alpha\gamma$, $p_2 = \alpha(1 - \gamma)$, $p_3 = \gamma(1 - \alpha)$, $p_4 = (1 - \alpha)(1 - \gamma)$ representa el máximo de entropía bajo estas restricciones.

Ejercicio 7.9 Un algoritmo probabilístico calcula la probabilidad p de que un correo electrónico entrante sea spam. Para clasificar los correos electrónicos en clases borrar y leer, luego se aplica una matriz de costos al resultado. (a) Proporcione una matriz de costos (matriz 2×2) para el filtro de spam. Suponga aquí que al usuario le cuesta 10 centavos eliminar un correo electrónico, mientras que la pérdida de un correo electrónico cuesta 10 dólares (compare esto con el Ejemplo 1.1 en la página 11 y el Ejercicio 1.7 en la página 14). (b) Demuestre que, para el caso de una matriz de 2×2 , la aplicación de la matriz de costos es equivalente a la aplicación de un umbral sobre la probabilidad de spam y determine el umbral.

Ejercicio 7.10 Dada una red bayesiana con las tres variables binarias A,B,C y $P(A) = 0.2$, $P(B) = 0.9$, así como el CPT que se muestra a continuación:

- (a) Calcule $P(A|B)$. (b) Calcule $P(C|A)$.

PAA (C) wf 0.1
ww 0.2
adelante 0.9
en adelante 0.4



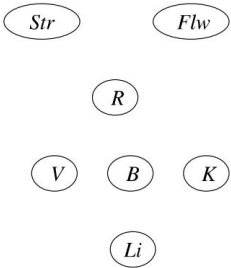
- Ejercicio 7.11 Para el ejemplo de alarma (Ejemplo 7.7 en la página 145), calcule las siguientes probabilidades condicionales: (a) Calcule las probabilidades a priori $P(AI)$, $P(J)$, $P(M)$. (b) Calcule $P(M|Bur)$ utilizando la regla del producto, la marginación, la regla de la cadena y independencia condicional. (c) Utilice la fórmula de Bayes para calcular $P(Bur|M)$. (d) Calcule $P(AI|J,M)$ y $P(Bur|J,M)$. (e) Demuestre que las variables J y M no son independientes. (f) Verifique todos sus resultados con JavaBayes y con PIT (vea [Ert11] para una demostración programas). (g) Diseñe una red bayesiana para el ejemplo de alarma, pero con la variable modificada que ordena M,AI,Ear,Bur,J . De acuerdo con la semántica de las redes bayesianas, solo se deben dibujar los bordes necesarios. (Pista: el orden de las variables dado aquí NO representa causalidad. Por lo tanto, será difícil determinar intuitivamente las independencias condicionales). (h) En la red bayesiana original del ejemplo de alarma, se eliminan los nodos del terremoto. ¿Qué CPT cambia esto? (¿Por qué estos en particular?) (i) Calcular el CPT del nodo de alarma en la nueva red.

Ejercicio 7.12 Se va a crear un sistema de diagnóstico para una luz de bicicleta accionada por dinamo utilizando una red bayesiana. Se dan las variables en la siguiente tabla.

Abr. Lo que significa		Valores
que Li Light está en		t/f
Calle	condición de calle	seco, mojado, cubierto de nieve
Volante motor Flw Dynamo desgastado t/f		
R	Dinamo deslizante t/f	
V	Dynamo muestra voltaje t/f	
B	Bombilla ok t/f	
k	Cable ok t/f	

Las siguientes variables son independientes por pares:
Calle, Flujo, B, K. Además: (R, B), (R, K), (V , B),
(V , K) son independientes y la siguiente ecuación
sostiene:

$$P(Li|V,R) = P(Li|V)$$
$$P(V|R,Estr) = P(V|R)$$
$$P(V|R,Flw) = P(V|R)$$



- (a) Dibuje todos los bordes en el gráfico (teniendo en cuenta la causalidad).
- (b) Ingrese todos los CPT faltantes en el gráfico (tabla de probabilidades condicionales). Inserte libremente valores plausibles para las probabilidades.
- (c) Demuestre que la red no contiene un borde (Str,Li).
- (d) Calcule $P(V|R = \text{cubierto de nieve})$.

VBK P(Li)
ttt 0.99
tff 0.01
t pies 0.01
t ff 0.001
pie t 0.3
pies f 0.005
fft 0.005
fff 0

Si definimos la IA como se hace en el libro de Elaine Rich [Ric83]:

La inteligencia artificial es el estudio de cómo hacer que las computadoras hagan cosas en las que, al momento, la gente es mejor.

y si consideramos que la capacidad de aprendizaje de la computadora es especialmente inferior a la de los humanos, entonces se deduce que la investigación de los mecanismos de aprendizaje y el desarrollo de algoritmos de aprendizaje automático es una de las ramas más importantes de IA.

También hay demanda de aprendizaje automático desde el punto de vista del software. desarrollador que programa, por ejemplo, el comportamiento de un robot autónomo. El La estructura del comportamiento inteligente puede volverse tan compleja que es muy difícil o incluso imposible de programar casi de manera óptima, incluso con lenguajes modernos de alto nivel como PROLOG y Python.¹ Incluso se utilizan algoritmos de aprendizaje automático hoy para programar robots de una manera similar a cómo aprenden los humanos (ver Cap. 10 o [BCDS08, RGH+06]), a menudo en una mezcla híbrida de ser programado y aprendido.

comportamiento

La tarea de este capítulo es describir los algoritmos de aprendizaje automático más importantes y sus aplicaciones. El tema se introducirá en esta sección, seguido por importantes algoritmos fundamentales de aprendizaje en las siguientes secciones. La teoría y la terminología se construirán en paralelo a esto. El capítulo se cerrará con un resumen. y descripción general de los diversos algoritmos y sus aplicaciones. Nos limitaremos en este capítulo a los algoritmos de aprendizaje supervisados y no supervisados. Como clase importante de algoritmos de aprendizaje, las redes neuronales se tratarán en el cap. 9. Debido a su lugar especial y papel importante para los robots autónomos, el refuerzo el aprendizaje también tendrá su propio capítulo dedicado (Cap. 10).

¿Qué es el aprendizaje? Aprender vocabulario de una lengua extranjera, o técnicas términos, o incluso memorizar un poema puede ser difícil para muchas personas. para computadoras,

¹Python es un lenguaje de secuencias de comandos moderno con una sintaxis muy legible, tipos de datos poderosos y bibliotecas estándar extensas, que se pueden usar para este fin.

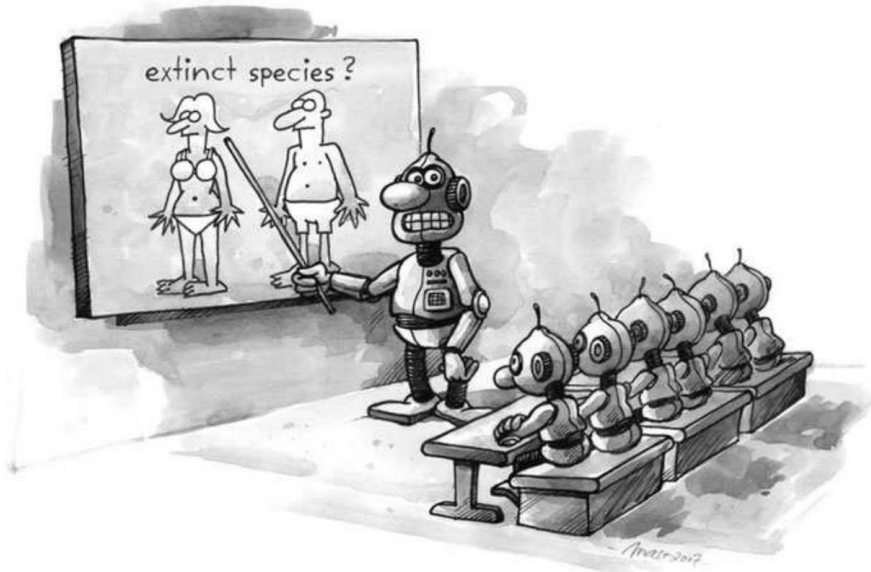


Fig. 8.1 Aprendizaje supervisado. . .

sin embargo, estas tareas son bastante simples porque son poco más que guardar texto en un archivo. Por lo tanto, la memorización no es interesante para la IA. Por el contrario, la adquisición de habilidades matemáticas no suele hacerse mediante la memorización. Para la suma de números naturales esto no es del todo posible, porque para cada uno de los términos en la suma $x + y$ hay infinitos valores. Para cada combinación de los dos valores x e y , habría que almacenar el triple $(x, y, x + y)$, lo cual es imposible. Para números decimales, esto es absolutamente imposible. Esto plantea la pregunta: ¿cómo aprendemos matemáticas? La respuesta dice: El maestro explica el proceso y los estudiantes lo practican en ejemplos hasta que ya no cometen errores en nuevos ejemplos. Después de 50 ejemplos, el estudiante (con suerte) entiende la suma. Es decir, después de solo 50 ejemplos puede aplicar lo aprendido a infinitos ejemplos nuevos, que hasta ese momento no se veían. Este proceso se conoce como generalización. Empezamos con un ejemplo sencillo.

Ejemplo 8.1 Un productor de frutas quiere dividir automáticamente las manzanas cosechadas en las clases de mercancías A y B. El dispositivo de clasificación está equipado con sensores para medir dos características, tamaño y color, y luego decidir a cuál de las dos clases pertenece la manzana. Esta es una tarea típica de clasificación. Los sistemas que son capaces de dividir vectores de características en un número finito de clases se denominan clasificadores.

Para configurar la máquina, las manzanas son recogidas a mano por un especialista, es decir, se clasifican. Luego, las dos medidas se ingresan junto con su etiqueta de clase en una tabla (Tabla 8.1 en la página 163). El tamaño viene dado en forma de diámetro en centímetros y el color por un valor numérico entre 0 (para verde) y 1 (para rojo).

Razonamiento con incertidumbre

7

Ya hemos mostrado en la Secc. 4 con el problema de Piolín de que la lógica de dos valores conduce a problemas en el razonamiento cotidiano. En este ejemplo, las afirmaciones Piolín es un pingüino, Los pingüinos son pájaros y Todos los pájaros pueden volar conducen a la inferencia (semánticamente incorrecta) que Piolín puede volar. La teoría de la probabilidad proporciona un lenguaje en el que podemos formalizar el enunciado Casi todas las aves pueden volar y realizar inferencias sobre él. La teoría de la probabilidad es un método probado que podemos usar aquí porque la incertidumbre acerca de si las aves pueden volar puede modelarse bien mediante un valor de probabilidad. Mostraremos que afirmaciones como el 99% de todas las aves pueden volar, junto con la lógica probabilística, conducen a inferencias correctas.

El razonamiento bajo incertidumbre con recursos limitados juega un papel importante en situaciones cotidianas y también en muchas aplicaciones técnicas de IA. En estas áreas los procesos heurísticos son muy importantes, como ya hemos discutido en el Cap. 6. Por ejemplo, usamos técnicas heurísticas cuando buscamos estacionamiento en el tráfico de la ciudad. Las heurísticas por sí solas a menudo no son suficientes, especialmente cuando se necesita una decisión rápida debido a un conocimiento incompleto, como se muestra en el siguiente ejemplo. Un peatón cruza la calle y un automóvil se acerca rápidamente. Para evitar un accidente grave, el peatón debe reaccionar rápidamente. No es capaz de preocuparse por la información completa sobre el estado del mundo, que necesitaría para los algoritmos de búsqueda discutidos en el Cap. 6. Por lo tanto, debe llegar a una decisión óptima bajo las restricciones dadas (poco tiempo y poco conocimiento potencialmente incierto). Si piensa demasiado, será peligroso. En esta y muchas situaciones similares (ver Fig. 7.1 en la página 114), se necesita un método para razonar con conocimiento incierto e incompleto.

Queremos investigar las diversas posibilidades de razonamiento bajo incertidumbre en un ejemplo de diagnóstico médico simple. Si un paciente experimenta dolor en la parte inferior derecha del abdomen y un recuento elevado de glóbulos blancos (leucocitos), esto genera la sospecha de que podría ser apendicitis. Modelamos esta relación usando lógica proposicional con la fórmula

$$\text{Dolor de estómago inferior derecho} \quad \text{Leucocitos} > 10000 \rightarrow \text{Apendicitis}$$



Fig. 7.1 "¡Simplemente sentémonos y pensemos qué hacer!"

Si entonces sabemos que

Dolor de estómago inferior derecho Leucocitos > 10000

es cierto, entonces podemos usar *modus ponens* para derivar apendicitis. Este modelo es claramente demasiado grueso. En 1976, Shortliffe y Buchanan reconocieron esto al construir su sistema médico experto MYCIN [Sho76]. Desarrollaron un cálculo utilizando los llamados factores de certeza, que permitían representar la certeza de hechos y reglas.

A una regla $A \rightarrow B$ se le asigna un factor de certeza β . La semántica de una regla $A \rightarrow \beta B$ se define mediante la probabilidad condicional $P(B|A) = \beta$. En el ejemplo anterior, la regla podría leerse

Dolor de estómago inferior derecho Leucocitos > 10000 $\rightarrow 0,6$ Apendicitis.

Para razonar con este tipo de fórmulas, utilizaron un cálculo para conectar los factores de las reglas. Resultó, sin embargo, que con este cálculo se podían derivar resultados inconsistentes.

Como se discutió en el Cap. 4, también hubo intentos de resolver este problema mediante el uso de lógica no monótona y lógica predeterminada, que, sin embargo, finalmente no tuvieron éxito. La teoría de Dempster-Schäfer asigna una función de creencia $Bel(A)$ a una proposición lógica A , cuyo valor da el grado de evidencia de la verdad de A . Pero incluso este formalismo tiene debilidades, que se muestran en [Pea88] usando una variante de El ejemplo de Piolín. Incluso la lógica difusa, que sobre todo tiene éxito en la teoría de control, demuestra debilidades considerables al razonar bajo incertidumbre en aplicaciones más complejas [Elk93].

Desde mediados de la década de 1980, la teoría de la probabilidad ha tenido cada vez más influencia en la IA [Pea88, Che85, Whi96, Jen01]. En el campo del razonamiento con redes bayesianas, o probabilidad subjetiva, se ha asegurado un lugar firme entre las técnicas de IA exitosas. En lugar de la implicación como se la conoce en lógica (implicación material), aquí se utiliza la probabilidad condicional, que modela significativamente mejor el razonamiento causal cotidiano. El razonamiento con probabilidad se beneficia enormemente del hecho de que la teoría de la probabilidad es una rama bien establecida de las matemáticas con cientos de años de antigüedad.

En este capítulo seleccionaremos un elegante, pero para un libro de instrucciones algo inusual, punto de entrada en este campo. Después de una breve introducción a los fundamentos más importantes que se necesitan aquí para razonar con probabilidad, comenzaremos con un ejemplo simple pero importante para razonar con conocimiento incierto e incompleto. De una manera bastante natural, casi convincente, seremos conducidos al método de máxima entropía (MaxEnt). A continuación mostraremos la utilidad de este método en la práctica utilizando el sistema médico experto LEXMED. Finalmente, presentaremos el ahora extendido razonamiento con redes bayesianas y mostraremos la relación entre los dos métodos.

7.1 Computación con probabilidades

El lector familiarizado con la teoría de la probabilidad puede saltarse esta sección. Para todos los demás, daremos un impulso rápido y recomendaremos algunos libros de texto apropiados, como [Ros09, FPP07].

La probabilidad es especialmente adecuada para modelar el razonamiento bajo incertidumbre. Una razón de esto es que las probabilidades son intuitivamente fáciles de interpretar, lo cual se puede ver en el siguiente ejemplo elemental.

Ejemplo 7.1 Para un solo lanzamiento de un dado de juego (experimento), la probabilidad del evento "sacar un seis" es igual a $1/6$, mientras que la probabilidad de que ocurra "sacar un número impar" es igual a $1/2$.

Definición 7.1 Sea Ω el conjunto finito de eventos para un experimento. Cada evento $\omega \in \Omega$ representa un posible resultado de la tirada. Si estos eventos $\omega_i \in \Omega$ se excluyen mutuamente, pero cubren todos los resultados posibles del intento, entonces se denominan eventos elementales.

Ejemplo 7.2 Para una sola tirada de un dado de juego

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

porque dos de estos eventos no pueden ocurrir simultáneamente. Sacar un número par $\{2, 4, 6\}$ por lo tanto, no es un evento elemental, ni es rodar un número menor que cinco $\{1, 2, 3, 4\}$ porque $\{2, 4, 6\} \cap \{1, 2, 3, 4\} = \{2, 4\} = \emptyset$.

Dados dos eventos A y B , $A \cup B$ también es un evento. Ω en sí mismo se denota el cierto evento, y el conjunto vacío $\emptyset$ el evento imposible.

A continuación, utilizaremos la notación lógica proposicional para operaciones con conjuntos. Es decir, para el conjunto $A \cap B$ escribimos $A \wedge B$. Esto no es solo una transformación sintáctica, más bien, también es semánticamente correcto porque la intersección de dos conjuntos se define como

$$x \in A \cap B \text{ si y solo si } x \in A \text{ y } x \in B.$$

Debido a que esta es la semántica de $A \wedge B$, podemos y usaremos esta notación. Esto es también cierto para las otras operaciones con conjuntos unión y complemento, y lo haremos, como se muestra en la siguiente tabla, use la notación de lógica proposicional para ellos también.

Notación de conjuntos	Lógica proposicional	Descripción
$A \cap B$	$A \wedge B$	Intersección/y
$A \cup B$	$A \vee B$	Unión/o
A^c	$\neg A$	Complemento/negación
Ω	w	Cierto evento/verdadero
$\emptyset$	F	Evento imposible/falso

Las variables utilizadas aquí (por ejemplo, A , B , etc.) se denominan variables aleatorias en la teoría de la probabilidad. Aquí solo usaremos variables aleatorias discretas con dominios finitos. La variable `face_number` para una tirada de dados es discreta con los valores 1, 2, 3, 4, 5, 6. La probabilidad de sacar un cinco o un seis es igual a 1/3. Esto puede ser descrito por

$$P(\text{cara\_número} \in \{5, 6\}) = P(\text{cara\_número} = 5 \vee \text{cara\_número} = 6) = 1/3.$$

Se supone que el concepto de probabilidad nos da una descripción tan objetiva como posible de nuestra "creencia" o "convicción" sobre el resultado de un experimento. Todo deben ser posibles números en el intervalo $[0, 1]$, donde 0 es la probabilidad de que evento imposible y 1 la probabilidad del evento cierto. Llegamos a esto de la siguiente definición.

Definición 7.2 Sea finito $\Omega = \{\omega_1, \omega_2, \dots, \omega_n\}$. No hay un evento elemental preferido, lo que significa que asumimos una simetría relacionada con la frecuencia con la que aparece cada evento elemental. La probabilidad $P(A)$ del evento A es entonces

$$P(A) = \frac{|A|}{|\Omega|} = \frac{\text{Número de casos favorables para } A}{\text{Número de casos posibles}}.$$

Se sigue inmediatamente que todo evento elemental tiene la probabilidad $1/|\Omega|$.

El requisito de que los eventos elementales tengan la misma probabilidad se denomina suposición de Laplace y las probabilidades calculadas de este modo se denominan probabilidades de Laplace. Esta definición llega a su límite cuando el número de eventos elementales se vuelve infinito.

Sin embargo, debido a que aquí solo estamos viendo espacios de eventos finitos, esto no presenta un problema. Para describir eventos usamos variables con el número apropiado de valores. Por ejemplo, una variable `eye_color` puede tomar los valores verde, azul, marrón. `eye_color = blue` entonces describe un evento porque estamos tratando con una proposición con los valores de verdad `true` o `false`.

Para variables binarias (booleanas), la variable en sí ya es una proposición. Aquí es suficiente, por ejemplo, escribir $P(\text{JohnCalls})$ en lugar de $P(\text{JohnCalls} = \text{true})$.

Ejemplo 7.3 Según esta definición, la probabilidad de que salga un número par es

$$P(\text{cara\_número} \in \{2, 4, 6\}) = \frac{|\{2, 4, 6\}|}{|\{1, 2, 3, 4, 5, 6\}|} = \frac{3}{6} = \frac{1}{2}.$$

Las siguientes reglas importantes se derivan directamente de la definición.

Teorema 7.1

1. $P(\Omega) = 1$.

2. $P(\emptyset) = 0$, lo que significa que el evento imposible tiene una probabilidad de 0.

3. Para los eventos A y B exclusivos por pares, es cierto que $P(A \cup B) = P(A) + P(B)$.

4. Para dos eventos complementarios A y $\neg A$ es cierto que $P(A) + P(\neg A) = 1$.

5. Para eventos arbitrarios A y B es cierto que $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

6. Para $A \subseteq B$ es cierto que $P(A) \leq P(B)$.

7. Si $A_1, \dots, A_n$ son eventos elementales, entonces $\sum_{i=1}^n P(A_i) = 1$ (normalización $i=1$ condición).

La expresión $P(A \cap B)$ o equivalentemente $P(A, B)$ representa la probabilidad de los eventos $A \cap B$. A menudo nos interesan las probabilidades de todos los elementos eventos, es decir, de todas las combinaciones de todos los valores de las variables A y B . Para el variables binarias A y B estas son $P(A, B)$, $P(A, \neg B)$, $P(\neg A, B)$, $P(\neg A, \neg B)$. Llamamos al vector

$$(P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

que consta de estos cuatro valores una distribución o distribución de probabilidad conjunta de los variables A y B . Una forma abreviada de esto es $P(A, B)$. La distribución en el caso de dos variables se pueden visualizar muy bien en forma de una tabla (matriz), representada como sigue:

PAG(A, B) B = w		segundo = f
A = w	PAG (A, B)	P (A, ¬B)
una = f	P(¬A,B) P (¬A,¬B)	

Para las d variables $X_1, \dots, X_d$ con n valores cada una, la distribución tiene los valores $P(X_1 = x_1, \dots, X_d = x_d)$ y $x_1, \dots, x_d$, cada uno de los cuales toma n valores diferentes. Por lo tanto, la distribución se puede representar como una matriz d -dimensional con un total de n^d elementos. Debido a la condición de normalización del Teorema 7.1 en la página 117, sin embargo, uno de estos n^d valores es redundante y la distribución se caracteriza por $n^d - 1$ valores únicos.

7.1.1 Probabilidad Condicional

Ejemplo 7.4 En la calle Landsdowne de Boston se mide la velocidad de 100 vehículos. Para cada medición también se anota si el conductor es un estudiante. El

los resultados son

Evento	Frecuencia Frecuencia relativa	
Vehículo observado	100	1
El conductor es un estudiante (S)	30	0.3
Velocidad demasiado alta (G)	10	0.1
El conductor es estudiante y va a exceso de velocidad (S ∩ G)	5	0.05

Planteamos la pregunta: ¿Los estudiantes aceleran con más frecuencia que el promedio por hijo, o que no estudiantes?

La respuesta está dada por la probabilidad

$$P(S|G) = \frac{| \text{El conductor es estudiante y va a exceso de velocidad} |}{| \text{El conductor es un estudiante} |} = \frac{5}{30} = \frac{1}{6} \approx 0,17$$

1Las probabilidades calculadas solo se pueden usar para proposiciones continuas si la muestra medida (100 vehículos) es representativa. De lo contrario, sólo se pueden hacer proposiciones sobre los 100 vehículos observados. hacerse.

por exceso de velocidad bajo la condición de que el conductor sea un estudiante. Obviamente, esto es diferente de la probabilidad a priori $P(G) = 0,1$ para el exceso de velocidad. Para la probabilidad a priori, el espacio del evento no está limitado por condiciones adicionales.

Definición 7.3 Para dos eventos A y B , la probabilidad $P(A|B)$ para A bajo la condición B (probabilidad condicional) está definida por

$$PAG(A|B) = \frac{PAG(A \cap B)}{PAG(B)}.$$

En el Ejemplo 7.4 vemos que en el caso de un espacio de eventos finito, la probabilidad condicional $P(A|B)$ puede entenderse como la probabilidad de $A \cap B$ cuando solo observamos el evento B , es decir, como

$$PAG(A|B) = \frac{|A \cap B|}{|B|}.$$

Esta fórmula se puede derivar fácilmente usando la Definición 7.2 en la página 117

$$PAG(A|B) = \frac{PAG(A \cap B)}{PAG(B)} = \frac{\frac{|A \cap B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{|A \cap B|}{|B|}.$$

Definición 7.4 Si, para dos eventos A y B ,

$$PAG(A|B) = PAG(A),$$

entonces estos eventos se llaman independientes.

Por lo tanto, A y B son independientes si la probabilidad del evento A no está influenciada por el evento B .

Teorema 7.2 Para eventos independientes A y B , se sigue de la definición que

$$PAG(A \cap B) = PAG(A) \cdot PAG(B).$$

Ejemplo 7.5 Para una tirada de dos dados, la probabilidad de sacar dos seises es $1/36$ si los dos dados son independientes porque

$$\text{PAG}(D1 = 6 \quad D2 = 6) = \text{PAG}(D1 = 6) \cdot \text{PAG}(D2 = 6) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36},$$

donde la primera ecuación solo es cierta cuando los dos dados son independientes. Si, por ejemplo, por algún poder mágico, el dado 2 es siempre el mismo que el dado 1, entonces

$$\text{PAG}(D1 = 6 \quad D2 = 6) = \frac{1}{6}.$$

Cadena de reglas

Resolver la definición de probabilidad condicional para $P(A|B)$ da como resultado la llamada regla del producto

$$\text{PAG}(A|B) = \frac{\text{PAG}(A \cap B)}{\text{PAG}(B)},$$

que generalizamos inmediatamente para el caso de n variables. Aplicando repetidamente la regla anterior obtenemos la regla de la cadena

$$\begin{aligned} P(X_1, \dots, X_n) &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_1, \dots, X_{n-1}) \\ &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_{n-1} | X_1, \dots, X_{n-2}) \cdot P(X_1, \dots, X_{n-2}) \\ &= P(X_n | X_1, \dots, X_{n-1}) \cdot P(X_{n-1} | X_1, \dots, X_{n-2}) \cdot \dots \cdot P(X_2 | X_1) \cdot P(X_1) \\ &= \prod_{i=1}^{n-1} P(X_i | X_1, \dots, X_{i-1}), \end{aligned} \quad (7.1)$$

con lo cual podemos representar una distribución como producto de probabilidades condicionales. Debido a que la regla de la cadena se cumple para todos los valores de las variables $X_1, \dots, X_n$, se formuló para la distribución usando el símbolo P .

marginación

Porque $A \cap (A \cap B) = (A \cap B)$ es cierto para las variables binarias A y B

$$\text{PAG}(A) = \text{PAG}((A \cap B) \cup (A \cap \neg B)) = \text{PAG}(A \cap B) + \text{PAG}(A \cap \neg B).$$

Mediante la suma de los dos valores de B , se elimina la variable B . Análogamente, para variables arbitrarias $X_1, \dots, X_d$, una variable, por ejemplo X_d , puede eliminarse sumando todas sus variables y obtenemos

$$P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}) = \sum_{x_d} P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}, X_d = x_d).$$

x_d

La aplicación de esta fórmula se denomina marginación. Esta suma puede continuar con las variables $X_1, \dots, X_{d-1}$ hasta que solo quede una variable. marginación también se puede aplicar a la distribución $P(X_1, \dots, X_d)$. La distribución resultante $P(X_1, \dots, X_{d-1})$ se denomina distribución marginal. Es comparable a la proyección de un paralelepípedo rectangular sobre una superficie plana. Aquí el objeto tridimensional es dibujado en el borde o "margen" del paralelepípedo, es decir, en un conjunto bidimensional. En ambos casos la dimensionalidad se reduce en uno.

Ejemplo 7.6 Observamos el conjunto de todos los pacientes que acuden al médico con dolor de estómago. Para cada paciente se mide el valor de leucocitos, que es una métrica por la abundancia relativa de glóbulos blancos en la sangre. Definimos la variable Leuko, que es cierto si y solo si el valor de leucocitos es superior a 10.000. Este indica una infección en el cuerpo. De lo contrario, definimos la variable App, que dice nosotros si el paciente tiene apendicitis, es decir, un apéndice infectado. La distribución $P(\text{App}, \text{Leuko})$ de estas dos variables se da en la siguiente tabla:

$P(\text{Aplicación}, \text{Leuko})$	Aplicación	$\neg$ Aplicación	Total
leuco	0,23	0,31	0.54
$\neg$ Leuko	0,05	0,41	0.46
Total	0,28	0,72	1

En la última fila se da la suma de las filas, y en la última columna la suma de las se dan columnas. A estas sumas se llega por marginación. por ejemplo, nosotros leer

$$P(\text{Leuko}) = P(\text{App}, \text{Leuko}) + P(\neg \text{App}, \text{Leuko}) = 0,54.$$

La distribución dada $P(\text{App}, \text{Leuko})$ podría provenir de una encuesta de médicos alemanes, Por ejemplo. A partir de ella podemos calcular la probabilidad condicional

$$P(\text{Leuko}|\text{Aplicación}) = \frac{P(\text{Leuko}, \text{aplicación})}{P(\text{aplicación})} = 0,82$$

lo que nos dice que alrededor del 82% de todos los casos de apendicitis conducen a un valor alto de leucocitos. Valores como este están publicados en la literatura médica. Sin embargo, la probabilidad condicional $P(\text{App}|\text{Leuko})$, que en realidad sería mucho más útil para diagnosticar apendicitis, no está publicado. Para entender esto, primero derivaremos un simple, pero fórmula muy importante.

Teorema de Bayes

Intercambiar A y B en la Definición 7.3 en la página 119 da como resultado

$$P(A|B) = \frac{P(A, B)}{P(B)} \quad \text{y} \quad P(B|A) = \frac{P(A, B)}{P(A)}.$$

Resolviendo las dos ecuaciones para $P(A|B)$ e igualándolas obtenemos el teorema de Bayes

$$PAG(A|B) = \frac{PAG(B|A) \cdot PAG(A)}{PAG(B)}, \quad (7.2)$$

que aplicamos inmediatamente al problema de la apendicitis y obtenemos

$$P(\text{Aplicación}|\text{Leuko}) = \frac{P(\text{Leuko}|\text{Aplicación}) \cdot P(\text{Aplicación})}{P(\text{Leuko})} = \frac{0,82 \cdot 0,28}{0,54} = 0,43. \quad (7.3)$$

Ahora bien, ¿por qué se publica $P(\text{Leuko}|\text{App})$, pero no $P(\text{App}|\text{Leuko})$?

Suponiendo que la apendicitis afecta al cuerpo humano independientemente de la raza, $P(\text{Leuko}|\text{App})$ es un valor universal que es cierto en todo el mundo. En (7.3), vemos que $P(\text{App}|\text{Leuko})$ no es universal porque este valor está influenciado por las habilidades probabilísticas a priori $P(\text{App})$ y $P(\text{Leuko})$. Cada uno de estos puede variar según las circunstancias de la vida. Por ejemplo, $P(\text{Leuko})$ depende de si hay muchas o pocas enfermedades infecciosas en una población. Este valor podría variar considerablemente entre los trópicos y las regiones más frías. Sin embargo, el teorema de Bayes simplifica el cálculo de $P(\text{App}|\text{Leuko})$, que es relevante para el diagnóstico, a partir del valor universalmente válido $P(\text{Leuko}|\text{App})$.

Antes de profundizar más en este ejemplo y expandirlo a un sistema médico experto para la apendicitis, primero debemos introducir la inferencia probabilística necesaria mecanismo.

7.2 El principio de máxima entropía

Ahora mostraremos, usando un ejemplo de inferencia, que un cálculo para razonar bajo incertidumbre se puede realizar usando la teoría de la probabilidad. Sin embargo, pronto veremos que los trillados caminos probabilísticos pronto llegan a su fin. Específicamente, cuando se dispone de muy poco conocimiento para resolver las ecuaciones necesarias, se necesitan nuevas ideas. El físico estadounidense ET Jaynes realizó un trabajo pionero en esta área en la década de 1950. Afirmó que dado el conocimiento faltante, uno puede maximizar la entropía de la distribución de probabilidad deseada y aplicó este principio a muchos ejemplos en [Jay57, Jay03]. Este principio luego se desarrolló aún más [Che83, Nil86, Kan89, KK92] y ahora está maduro y se puede aplicar tecnológicamente, lo que mostraremos en el ejemplo del proyecto LEXMED en la Secc. 7.3.

7.2.1 Una regla de inferencia para probabilidades

Queremos derivar una regla de inferencia para el conocimiento incierto que sea análoga al modus ponens. A partir del conocimiento de una proposición A y una regla A $\rightarrow$ B, se llegará a la conclusión B. Formulado sucintamente, esto dice

$$\frac{A, A \rightarrow B}{B}.$$

La generalización para las reglas de probabilidad produce

$$\frac{P(A) = \alpha, P(B|A) = \beta P(B)}{=?}.$$

Se dan las dos reglas de probabilidad α , β y se desea el valor $P(B)$. Por marginalización obtenemos la distribución marginal deseada

$$P(B) = P(A, B) + P(\neg A, B) = P(B|A) \cdot P(A) + P(B|\neg A) \cdot P(\neg A).$$

Se conocen los tres valores $P(A)$, $P(\neg A)$, $P(B|A)$ del lado derecho, pero se desconoce el valor $P(B|\neg A)$. No podemos hacer una afirmación exacta sobre $P(B)$ con la teoría clásica de la probabilidad, pero como mucho podemos estimar $P(B) \geq P(B|A) \cdot P(A)$.

Consideremos ahora la distribución

$$P(A, B) = (P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

e introducir de forma abreviada las cuatro incógnitas

$$\begin{aligned} p_1 &= P(A, B), \\ p_2 &= P(A, \neg B), \quad p_3 \\ &= P(\neg A, B), \\ p_4 &= P(\neg A, \neg B). \end{aligned}$$

Estos cuatro parámetros determinan la distribución. Si se conocen todas, se pueden calcular todas las probabilidades de las dos variables A y B. Para calcular las cuatro incógnitas, se necesitan cuatro ecuaciones. Ya se conoce una ecuación en forma de condición de normalización

$$p_1 + p_2 + p_3 + p_4 = 1.$$

Por lo tanto, se necesitan tres ecuaciones más. En nuestro ejemplo, sin embargo, solo se conocen dos ecuaciones.

A partir de los valores dados $P(A) = \alpha$ y $P(B|A) = \beta$ calculamos

$$P(A, B) = P(B|A) \cdot P(A) = \alpha\beta$$

y

$$P(A) = P(A, B) + P(A, \neg B).$$

A partir de esto podemos establecer el siguiente sistema de ecuaciones y resolverlo en la medida de lo posible:

$$p_1 = \alpha\beta, \quad (7.4)$$

$$p_1 + p_2 = \alpha, \quad (7.5)$$

$$p_1 + p_2 + p_3 + p_4 = 1, \quad (7.6)$$

$$(7.4) \text{ en } (7.5): \quad p_2 = \alpha - \alpha\beta = \alpha(1 - \beta), \quad (7.7)$$

$$(7.5) \text{ en } (7.6): \quad p_3 + p_4 = 1 - \alpha. \quad (7.8)$$

Las probabilidades p_1 , p_2 para las interpretaciones (A, B) y $(A, \neg B)$ son conocidas, pero para los valores p_3 , p_4 solo queda una ecuación. Para llegar a una solución definitiva a pesar de este conocimiento faltante, cambiamos nuestro punto de vista. Usamos la ecuación dada como una restricción para la solución de un problema de optimización.

Estamos buscando una distribución p (para las variables p_3 , p_4) que maximice la entropía

$$H(p) = -\sum_{i=1}^n p_i \ln p_i = -p_3 \ln p_3 - p_4 \ln p_4 \quad (7.9)$$

bajo la restricción $p_3 + p_4 = 1 - \alpha$ (7.8). ¿Por qué exactamente debería maximizarse la función de entropía? Debido a que nos falta información sobre la distribución, de alguna manera debe agregarse. Podríamos fijar un valor ad hoc, por ejemplo, $p_3 = 0.1$. Sin embargo, es mejor determinar los valores p_3 y p_4 de forma que la información añadida sea mínima. Podemos demostrar (Sección 8.4.2 y [SW76]) que la entropía mide la incertidumbre de una distribución hasta un factor constante. La entropía negativa es entonces una medida de la cantidad de información que contiene una distribución. La maximización de la entropía minimiza el contenido de información de la distribución. Para visualizar esto, la función de entropía para el caso bidimensional se representa gráficamente en la Fig. 7.2 en la página 125.

Para determinar el máximo de la entropía bajo la restricción $p_3 + p_4 - 1 + \alpha = 0$ usamos el método de los multiplicadores de Lagrange [Ste07]. La función de Lagrange dice

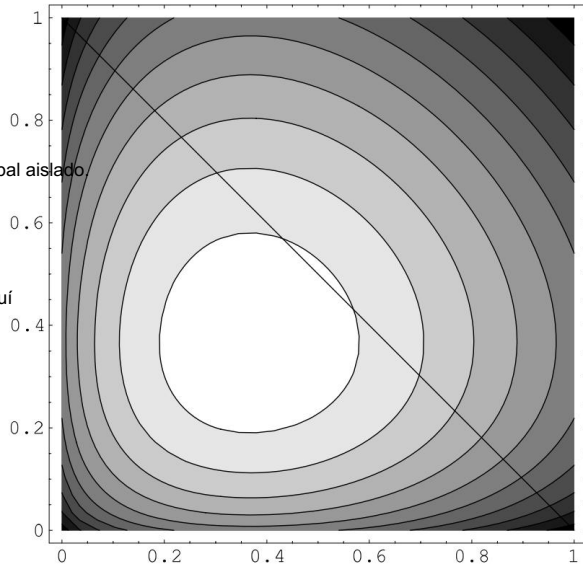
$$L = -p_3 \ln p_3 - p_4 \ln p_4 + \lambda(p_3 + p_4 - 1 + \alpha).$$

Tomando las derivadas parciales con respecto a p_3 y p_4 obtenemos

$$\begin{aligned} \frac{\partial L}{\partial p_3} &= -\ln p_3 - 1 + \lambda = 0, \quad \partial p_3 \\ &= \\ \frac{\partial L}{\partial p_4} &= -\ln p_4 - 1 + \lambda = 0, \quad \partial p_4 \end{aligned}$$

Fig. 7.2 Diagrama de líneas de contorno de la función de entropía bidimensional.

Vemos que es estrictamente convexo en todo el cuadrado unitario y que tiene un máximo global aislado. También se marca la restricción $p_3 + p_4 = 1$ como un caso especial de la condición $p_3 + p_4 = 1 + \alpha = 0$ para $\alpha = 0$ que es relevante aquí



y calcular

$$p_3 = p_4 = \frac{1 - \alpha}{2}.$$

Ahora podemos calcular el valor deseado.

$$P(B) = P(A, B) + P(\neg A, B) = p_1 + p_3 = \alpha\beta + \alpha\beta - 2 \frac{1 - \alpha}{2} = \alpha\beta - \frac{1 - \alpha}{2}.$$

Sustituyendo en α y β se obtiene

$$PAGS(B) = PAGS(A) PAGS(B|A) - \frac{1}{2} \left(1 - \frac{1}{2} \right).$$

$P(B)$ se muestra en la Fig. 7.3 en la página 126 para varios valores de $P(B|A)$. Vemos que en el caso de borde de dos valores, es decir, cuando $P(B)$ y $P(B|A)$ toman los valores 0 o 1, la inferencia probabilística devuelve el mismo valor para $P(B)$ que el modus ponens.

Cuando A y $B|A$ son verdaderos, B también lo es. Un caso interesante es $P(A) = 0$, en el que $\neg A$ es verdadero. Modus ponens no se puede aplicar aquí, pero nuestra fórmula da como resultado el valor $1/2$ para $P(B)$ independientemente de $P(B|A)$. Cuando A es falso, no sabemos nada acerca de B , lo que refleja exactamente nuestra intuición. El caso donde $P(A) = 1$ y $P(B|A) = 0$ también está cubierto por la lógica proposicional. Aquí A es verdadero y $A \rightarrow B$ falso, y por lo tanto $A \rightarrow \neg B$ verdadero. Entonces B es falso. La línea horizontal en la figura significa que no podemos hacer una predicción sobre B en el caso de $P(B|A) = 1/2$. Entre estos puntos, $P(B)$ cambia linealmente para cambios en $P(A)$ o $P(B|A)$.

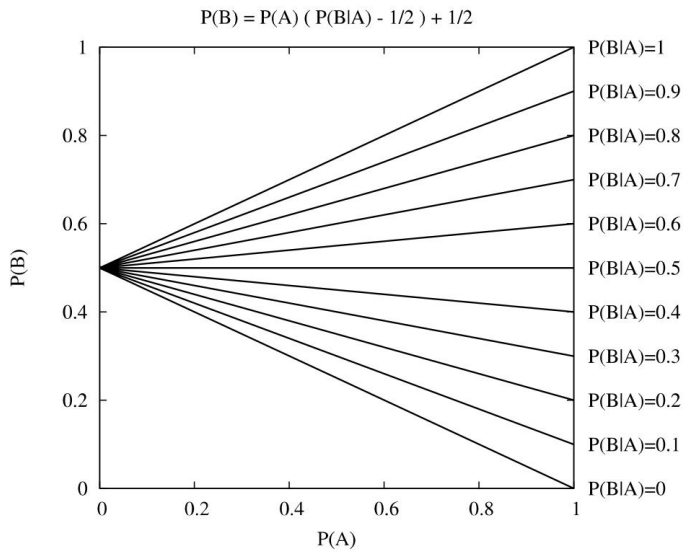


Fig. 7.3 Conjunto de curvas para $P(B)$ en función de $P(A)$ para diferentes valores de $P(B|A)$

Teorema 7.3 Sea un conjunto consistente² de ecuaciones probabilísticas lineales. Entonces existe un máximo único para la función de entropía con las ecuaciones dadas como restricciones. La distribución MaxEnt así definida tiene un contenido mínimo de información bajo las restricciones.

De este teorema se sigue que no hay distribución que satisfaga las restricciones y tenga una entropía más alta que la distribución MaxEnt. Un cálculo que conduce a una entropía más baja incluye información ad hoc adicional, que no está justificada.

Mirando más de cerca el cálculo anterior de $P(B)$, vemos que los dos valores p_3 y p_4 siempre ocurren simétricamente. Esto significa que intercambiar las dos variables no cambia el resultado. Por tanto, el resultado final es $p_3 = p_4$. La llamada indiferencia de estas dos variables hace que MaxEnt las iguale. Esta relación es válida generalmente:

Definición 7.5 Si un intercambio arbitrario de dos o más variables en las ecuaciones de La Grange da como resultado ecuaciones equivalentes, estas variables se denominan indiferentes.

²Se dice que un conjunto de ecuaciones probabilísticas es consistente si tiene al menos una solución, es decir, una distribución que satisface todas las ecuaciones.

Teorema 7.4 Si un conjunto de variables $\{p_1, \dots, p_k\}$ es indiferente, entonces el máximo de la entropía bajo las restricciones dadas está en el punto donde $p_1 = p_2 = \dots = p_k$.

Con este conocimiento podríamos haber igualado inmediatamente las dos variables p_3 y p_4 (sin resolver las ecuaciones de Lagrange).

7.2.2 Entropía máxima sin restricciones explícitas

Veamos ahora el caso en el que no se da ningún conocimiento. Esto significa que, además de la condición de normalización

$$p_1 + p_2 + \dots + p_n = 1,$$

no hay restricciones. Por lo tanto, todas las variables son indiferentes. Por lo tanto podemos ³

igualémoslos y se sigue que $p_1 = p_2 = \dots = p_n = 1/n$.

Para razonar bajo

incertidumbre, esto significa que dada una completa falta de conocimiento, todos los mundos son igualmente probables. Es decir, la distribución es uniforme. Por ejemplo, en el caso de dos variables A y B se daría el caso de que

$$P(A, B) = P(A, \neg B) = P(\neg A, B) = P(\neg A, \neg B) = 1/4,$$

de donde se siguen $P(A) = P(B) = 1/2$ y $P(B|A) = 1/2$. El resultado para el caso bidimensional se puede ver en la Fig. 7.2 en la página 125 porque la condición marcada es exactamente la condición de normalización. Vemos que el máximo de la entropía se encuentra exactamente en la línea $(1/2, 1/2)$.

Tan pronto como el valor de una condición se desvía del derivado de la distribución uniforme, las probabilidades de los mundos cambian. Mostramos esto en otro ejemplo. Con las mismas descripciones que se usaron anteriormente, asumimos que solo

$$P(B|A) = \beta$$

es conocida. Así $P(A, B) = P(B|A)P(A) = \beta P(A)$, de donde se sigue $p_1 = \beta(p_1 + p_2)$ y derivamos las dos restricciones

$$\beta p_2 + (\beta - 1)p_1 = 0,$$

$$p_1 + p_2 + p_3 + p_4 - 1 = 0.$$

³El lector puede calcular este resultado mediante la maximización de la entropía bajo la condición de normalización (Ejercicio 7.5 en la página 158).

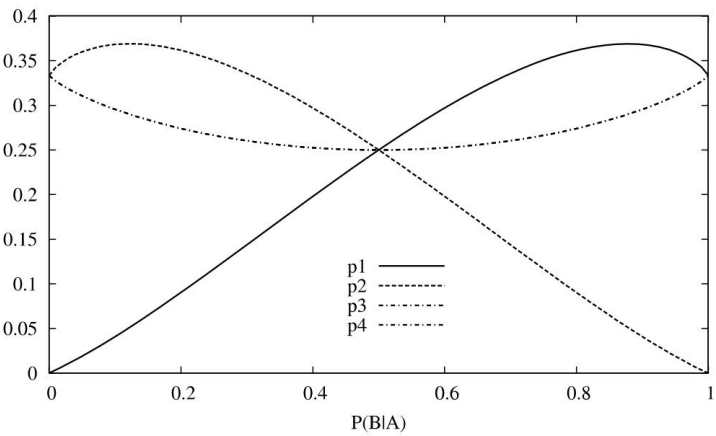


Fig. 7.4 p1, p2, p3, p4 en dependencia de β

Tabla 7.1 Tabla de verdad para implicaciones materiales y probabilidad condicional de límite de la lógica proposicional

ABA	B	PAG (A)	PAG (B)	PAG (B A)
ttt	111			
ff	100			
pies w	pies 0	0	1	Indefinido
			0	Indefinido

Aquí las ecuaciones de Lagrange ya no pueden resolverse simbólicamente tan fácilmente. Una solución numérica de las ecuaciones de Lagrange produce la imagen representada en la figura 7.4, lo que muestra que $p3 = p4$. Ya podemos ver esto en las restricciones, en las que $p3$ y $p4$ son indiferentes. Para $P(B|A) = 1/2$ obtenemos la distribución uniforme, que no es una sorpresa. Esto significa que la restricción de este valor no implica una restricción en la distribución. Además, podemos ver que para $P(B|A)$ pequeño, $P(A, B)$ es también pequeño.

7.2.3 Probabilidad condicional versus implicación material

Ahora mostraremos que, para modelar el razonamiento, la probabilidad condicional es mejor que lo que se conoce en lógica como implicación material (a este fin, ver también [Ada75]). Primero observamos la tabla de verdad que se muestra en la Tabla 7.1, en la que la probabilidad condicional y la implicación material para los casos extremos de probabilidades cero y uno son comparados. En ambos casos con premisas falsas (que, intuitivamente, son casos críticos), $P(B|A)$ no está definido, lo que tiene sentido.

Ahora nos preguntamos qué valor toma $P(B|A)$ cuando se dan valores arbitrarios de $P(A) = \alpha$ y $P(B) = \gamma$ y no se conoce otra información. Nuevamente maximizamos la entropía bajo las restricciones dadas. Como arriba establecemos

$$p_1 = P(A, B), p_2 = P(A, \neg B), p_3 = P(\neg A, B), p_4 = P(\neg A, \neg B)$$

y obtener como restricciones

$$p_1 + p_2 = \alpha, \quad (7.10)$$

$$p_1 + p_3 = \gamma, \quad (7.11)$$

$$p_1 + p_2 + p_3 + p_4 = 1. \quad (7.12)$$

Con esto calculamos utilizando la maximización de entropía (ver Ejercicio 7.8 en la página 159)

$$p_1 = \alpha\gamma, p_2 = \alpha(1 - \gamma), p_3 = \gamma(1 - \alpha), p_4 = (1 - \alpha)(1 - \gamma).$$

De $p_1 = \alpha\gamma$ se sigue que $P(A, B) = P(A) \cdot P(B)$, lo que significa que A y B son independientes. Debido a que no hay restricciones que conecten A y B, el principio de MaxEnt da como resultado la independencia de estas variables. La mitad derecha de la Tabla 7.1 en la página 128 hace que esto sea más fácil de entender. De la definición

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

se sigue para el caso $P(A) = 0$, es decir, cuando la premisa no es falsa, porque A y B son independientes, que $P(B|A) = P(B)$. Para el caso $P(A) = 0$, $P(B|A)$ permanece sin definir.

7.2.4 Sistemas MaxEnt

Como se mencionó anteriormente, debido a la no linealidad de la función de entropía, la optimización de MaxEnt generalmente no se puede realizar simbólicamente para problemas no triviales.

Así, se desarrollaron dos sistemas para la maximización de la entropía numérica. El primer sistema, SPIRIT (Symmetrical Probabilistic Intensional Reasoning in Inference Networks in Transition, www.xspirit.de), [RM96] se construyó en Fernuniversität Hagen.

El segundo, PIT (herramienta de inducción de probabilidad) fue desarrollado en la Universidad Técnica de Munich [Sch96, ES99, SE00]. Ahora presentaremos brevemente PIT.

El sistema PIT utiliza el método de programación cuadrática secuencial (SQP) para encontrar un extremo de la función de entropía bajo las restricciones dadas. Como entrada, PIT espera datos que contengan las restricciones. Por ejemplo, las restricciones $P(A) = \alpha$ y $P(B|A) = \beta$ de la Secc. 7.2.1 tiene la forma

```
var A{t,f}, B{t,f};

P([A=t]) = 0,6;
P([B=t] | [A=t]) = 0,3;

QP([B=t]);
QP([B=t] | [A=t]);
```

Debido a que PIT realiza un cálculo numérico, tenemos que ingresar valores de probabilidad explícitos. La penúltima fila contiene la consulta $QP([B=t])$. Esto significa que $P(B)$ es el valor deseado. En www.pit-systems.de bajo "Ejemplos" ahora coloque esta entrada en una página de entrada en blanco ("Página en blanco") e inicie PIT. como resultado nosotros conseguir

No. Valor de verdad	Consulta de probabilidad
1 SIN ESPECIFICAR	$3.800e-01 \text{ } QP([B = t]);$
2 SIN ESPECIFICAR	$3.000e-01 \text{ } QP([A = t] \text{ } \vdash \text{ } [B = t]);$

ya partir de ahí leer $P(B) = 0,38$ y $P(B|A) = 0,3$.

7.2.5 El ejemplo de Piolín

Ahora mostramos, usando el ejemplo de Piolín de la Secta. 4.3, que el razonamiento probabilístico y en particular MaxEnt no son monótonos y modelan el razonamiento cotidiano muy Bueno. Modelamos las reglas relevantes con probabilidades de la siguiente manera:

$P(\text{pájaro} \text{pingüino}) = 1$	"los pingüinos son pájaros"
$(\text{moscas} \text{pájaro}) \quad [0.95, 1]$	"(casi todas) las aves pueden volar"
$P(\text{moscas} \text{pingüino}) = 0$	"los pingüinos no pueden volar"

La primera y la tercera reglas representan predicciones firmes, que también pueden formularse fácilmente en lógica. En el segundo, sin embargo, expresamos nuestro conocimiento de que casi todos las aves pueden volar por medio de un intervalo de probabilidad. Con los datos de entrada PIT

```
var pingüino{sí,no}, pájaro{sí,no}, moscas{sí,no};

P([pájaro=sí] | [pingüino=sí]) = 1;
P([moscas=sí] | [pájaro=sí]) IN [0.95,1];
P([moscas=sí] | [pingüino=sí]) = 0;

QP([moscas=sí] | [pingüino=sí]);
```

obtenemos la respuesta correcta

No. Valor de verdad	Consulta de probabilidad
1 SIN ESPECIFICAR	0.000e+00 QP([pingüino=sí]-> [moscas=sí]);

con la proposición de que los pingüinos no pueden volar.⁴ La explicación de esto es muy simple. Con $P(\text{moscas}|\text{pájaro}) \in [0.95, 1]$ es posible que haya pájaros que no vuelen. Si esto fueron reemplazadas por $P(\text{moscas}|\text{pájaro}) = 1$, entonces PIT no podría hacer nada y generaría un mensaje de error sobre restricciones inconsistentes.

En este ejemplo, podemos ver fácilmente que los intervalos de probabilidad suelen ser muy útiles para modelar nuestra ignorancia sobre valores de probabilidad exactos. Podríamos haber hecho un formulación aún más confusa de la segunda regla en el espíritu de "normalmente los pájaros vuelan" con $P(\text{moscas}|\text{pájaro}) \in (0.5, 1]$. El uso del intervalo semiabierto excluye el valor 0.5.

Ya se ha demostrado en [Pea88] que este ejemplo se puede resolver usando lógica probabilística, incluso sin MaxEnt. En [Sch96] se muestra para una serie de puntos de referencia exigentes para el razonamiento no monótono que estos pueden resolverse elegantemente con MaxEnt. En la siguiente sección presentamos una aplicación práctica exitosa de MaxEnt en forma de sistema médico experto.

7.3 LEXMED, un Sistema Experto para el Diagnóstico de Apendicitis

El sistema médico experto LEXMED, que utiliza el método MaxEnt, fue desarrollado en la Universidad de Ciencias Aplicadas de Ravensburg-Weingarten por Manfred Schramm, Walter Rampf y el autor, en cooperación con Weingarten 14-Nothelfer Hospital [SE00, Le999].⁵ El acrónimo LEXMED significa sistema experto de aprendizaje para el diagnóstico médico.

7.3.1 Diagnóstico de apendicitis con métodos formales

La causa grave más frecuente de dolor abdominal agudo [dD91] es la apendicitis: una inflamación del apéndice, un tubo con un extremo ciego conectado al ciego. Incluso hoy en día, el diagnóstico puede ser difícil en muchos casos [OFY+95]. Por ejemplo, hasta aproximadamente El 20% de los apéndices extirpados no tienen hallazgos patológicos, lo que significa que las operaciones eran innecesarias. Asimismo, se dan regularmente casos en los que un apéndice inflamado no se reconoce como tal.

Desde principios de la década de 1970, ha habido intentos de automatizar el diagnóstico de apendicitis, con el objetivo de reducir la tasa de diagnósticos falsos [dDLS+72, OPB94, OFY +95]. Destaca especialmente el sistema experto

⁴ $4QP([penguin=yes]-> [flies=yes])$ es una forma alternativa de la sintaxis PIT para $QP([moscas=sí] \mid [pingüino=sí])$.
⁵El proyecto fue financiado por el estado alemán de Baden-Württemberg, la compañía de seguros de salud AOK Baden-Württemberg, la Universidad de Ciencias Aplicadas de Ravensburg-Weingarten y el Hospital 14 Nothelfer en Weingarten.

para el diagnóstico de dolor abdominal agudo, desarrollado por de Dombal en Gran Bretaña. Se hizo público en 1972, claramente anterior al famoso sistema MYCIN.

Casi todos los procesos de diagnóstico formales utilizados en medicina hasta la fecha se han basado en puntajes. Los sistemas de puntuación son extremadamente fáciles de aplicar: para cada valor de un síntoma (por ejemplo, fiebre o dolor de estómago en la parte inferior derecha del estómago), el médico anota una determinada cantidad de puntos. Si la suma de los puntos supera un determinado valor (umbral), se recomienda una determinada decisión (por ejemplo, la operación). Para n síntomas $S_1, \dots, S_n$, una puntuación para apendicitis se puede describir formalmente como

$$\text{Diagnosticar} = \begin{matrix} \text{Apendicitis si } w_1S_1 + \dots + w_nS_n > \Theta, \text{ negativo} \\ \text{demás.} \end{matrix}$$

Con puntuaciones, una combinación lineal de valores de síntomas se compara así con un umbral Θ . Los pesos de los síntomas se extraen de bases de datos mediante métodos estadísticos. La ventaja de las partituras es su sencillez de aplicación. La suma ponderada de los puntos se puede calcular fácilmente a mano y no se necesita una computadora para el diagnóstico.

Debido a la linealidad de este método, las puntuaciones son demasiado débiles para modelar relaciones complejas. Dado que la contribución w_iS_i de un síntoma S_i a la puntuación se calcula independientemente de los demás síntomas, es evidente que los sistemas de puntuación no pueden tener en cuenta ningún "contexto". Principalmente, no pueden distinguir entre combinaciones de síntomas, por ejemplo, no pueden distinguir entre el recuento de glóbulos blancos de un paciente anciano y el de un paciente joven.

Para un conjunto dado fijo de síntomas, la probabilidad condicional es mucho más poderosa que las puntuaciones para hacer predicciones porque estas últimas no pueden describir las dependencias entre diferentes síntomas. Podemos mostrar que las puntuaciones asumen implícitamente que todos los síntomas son independientes.

Cuando se utilizan partituras, surge otro problema. Para llegar a una buena calidad de diagnóstico, debemos poner requisitos estrictos a las bases de datos utilizadas para determinar estadísticamente los pesos w_i .

En particular, deben ser representativos del conjunto de pacientes del área en la que se utiliza el sistema de diagnóstico. Esto es a menudo difícil, si no imposible, de garantizar. En tales casos, los puntajes y otros métodos estadísticos no se pueden usar o tendrán una alta tasa de errores.

7.3.2 Base de conocimiento probabilística híbrida

Las relaciones probabilísticas complejas aparecen con frecuencia en medicina. Con LEXMED, estas relaciones se pueden modelar bien y calcular rápidamente. Aquí es fundamental el uso de proposiciones probabilísticas, con las que se puede expresar y procesar información incierta e incompleta de forma intuitiva y matemáticamente fundamentada. La siguiente pregunta puede servir como una consulta típica contra el sistema experto: "¿Qué tan alta es la probabilidad de un apéndice inflamado si el paciente es un hombre de 23 años con dolor en el abdomen inferior derecho y un recuento de glóbulos blancos de 13.000?" Formulada como probabilidad condicional, usando los nombres y rangos de valores para los síntomas usados en la Tabla 7.2 en la página 133, esto dice

Tabla 7.2 Síntomas utilizados para la consulta en LEXMED y sus valores. El número de valores para cada síntoma se da en la columna marcada #

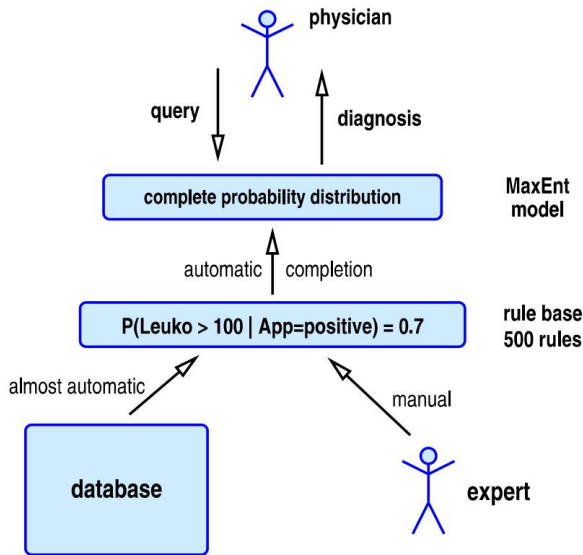
Síntoma	Valores	# Corto
Género	hombre, mujer	2 Sexo2
Edad	0–5, 6–10, 11–15, 16–20, 21–25, 26–35, 36–45, 46–55, 56–65, 65–	10 años10
Dolor 1er Quad.	si, no	2 P1Q2
Dolor 2º Cuad.	si, no	2 P2Q2
Dolor 3er Quad.	si, no	2 P3Q2
Dolor 4º Cuad.	si, no	2 P4Q2
protegiendo	Local, global, ninguno	3 Gua3
Sensibilidad de rebote	si, no	2 Teb2
Dolor al tocar	si, no	2 toque2
dolor rectal	si, no	2 RecP2
Sonidos intestinales	Débil, normal, aumentado, ninguno	4 ArcoS4
Ultrasonido anormal	si, no	2 Sono2
Sedim de orina anormal.	si, no	2 orina2
Temperatura (rectal)	–37,3, 37,4–37,6, 37,7–38,0, 38,1–38,4, 38,5–38,9, 39,0–	6 TRec6
leucocitos	0–6k, 6k–8k, 8k–10k, 10k–12k, 12k–15k, 15k–20k, 20k–	7 leuco7
Diagnóstico	Inflamado, perforado, negativo, otro	4 Diag4

P (Diag4 = inflamado Diag4 = perforado |
Sexo2 = hombre Edad10 21–25 Leuko7 12k–15k).

Mediante el uso de proposiciones probabilísticas, LEXMED tiene la capacidad de utilizar información de bases de datos no representativas porque esta información se puede complementar apropiadamente de otras fuentes. LEXMED subyacente es una base de datos que sólo contiene datos sobre pacientes cuyos apéndices fueron extirpados quirúrgicamente. Con los métodos estadísticos se generan (alrededor de 400) reglas que compilan el conocimiento contenido en la base de datos en forma abstracta[ES99]. porque no hay pacientes en esta base de datos que tenían sospecha de tener apendicitis pero tenían diagnósticos negativos (es decir, no requerían tratamiento),6 no hay conocimiento sobre pacientes en la base de datos. Por lo tanto, se debe agregar el conocimiento de otras fuentes. En LEXMED, por lo tanto, las reglas recopiladas de la base de datos se complementan con (alrededor de 100) reglas de expertos médicos y la literatura médica. Esto resulta en una base de datos probabilística híbrida, que contiene conocimiento extraído de datos como así como conocimientos formulados explícitamente por expertos. Debido a que ambos tipos de reglas son

6Estos diagnósticos negativos se denominan “dolor abdominal inespecífico” (NSAP).

Fig. 7.5 Las reglas probabilísticas se generan a partir de datos y conocimiento experto, que se integran en una base de reglas (base de conocimiento) y finalmente se completan utilizando el método MaxEnt.



formuladas como probabilidades condicionales (ver por ejemplo (7.14) en la página 138), se pueden combinar fácilmente, como se muestra en la Fig. 7.5 y con más detalles en la Fig. 7.7 en la página 136.

LEXMED calcula las probabilidades de varios diagnósticos utilizando la distribución de probabilidad de todas las variables relevantes (consulte la Tabla 7.2 en la página 133). Debido a que los 14 síntomas utilizados en LEXMED y los diagnósticos se modelan como variables discretas (incluso las variables continuas, como el valor de leucocitos, se dividen en rangos), el tamaño de la distribución (es decir, el tamaño del espacio de eventos) se puede determinar utilizando la Tabla 7.2 en la página 133 como el producto del número de valores de todos los síntomas, o

$$10 \cdot 2 \cdot 10 \cdot 3 \cdot 4 \cdot 6 \cdot 7 \cdot 4 = 20\,643\,840$$

elementos. Debido a la condición de normalización del Teorema 7.1, tiene 20 643 839 valores independientes. Cada conjunto de reglas con menos de 20 643 839 valores de probabilidad potencialmente no describe completamente este espacio de eventos. Para poder responder a cualquier consulta arbitraria, el sistema experto necesita una distribución completa. La construcción de una distribución tan extensa y consistente usando métodos estadísticos es muy difícil.⁷ Requerir de un experto humano todos los 20 643 839 valores para la distribución (en lugar de las 100 reglas antes mencionadas) sería esencialmente imposible.

Aquí entra en juego el método MaxEnt. La generalización de unas 500 reglas a un modelo de probabilidad completo se realiza en LEXMED maximizando la entropía con las 500 reglas como restricciones. Una codificación eficiente de la distribución MaxEnt resultante conduce a tiempos de respuesta para el diagnóstico de alrededor de un segundo.

⁷La tarea de generar una función a partir de un conjunto de datos se conoce como aprendizaje automático. Cubriremos esto a fondo en el Cap. 8.

Personal Details	unknown	values	
Gender	☒	☐ male ☐ female	?
Age-group	☐	☐ 0-5 ☐ 6-10 ☐ 11-15 ☐ 16-20 ☒ 21-25 ☐ 26-35 ☐ 36-45 ☐ 46-55 ☐ 56-65 ☐ 65-	?
Results of examination	not done	values	
1st quadrant	☒	☐ yes ☐ no	?
2nd quadrant	☒	☐ yes ☐ no	?
3rd quadrant	☐	☒ yes ☐ no	?
4th quadrant	☒	☐ yes ☐ no	?
guarding	☐	☒ local ☐ global ☐ none	?
rebound tenderness	☐	☒ yes ☐ no	?
pain on tapping	☐	☒ yes ☐ no	?
rectal pain	☒	☐ yes ☐ no	?
bowel sounds	☐	☐ weak ☒ normal ☐ increased ☐ none	?
abnormal ultrasound	☒	☐ yes ☐ no	?
abnormal urine sediment	☒	☐ yes ☐ no	?
temperature range (rectal)	☐	☐ -37.3 ☐ 37.4-37.6 ☐ 37.7-38.0 ☐ 38.1-38.4 ☒ 38.5-38.9 ☐ 39.0-	?
leucocyte count	☐	☐ 0-6k ☐ 6k-8k ☐ 8k-10k ☐ 10k-12k ☒ 12k-15k ☐ 15k-20k ☐ 20k-	?

Result of the PIT diagnosis				
Diagnosis	App. inflamed	App. perforated	Negative	Other
Probability	0.70	0.17	0.06	0.07

Fig. 7.6 La máscara de entrada LEXMED para la entrada de los síntomas examinados y debajo de ella la salida de las probabilidades de diagnóstico resultantes

7.3.3 Aplicación de LEXMED

El uso de LEXMED es simple y se explica por sí mismo. El médico visita el Para un diagnóstico
Página de inicio de LEXMED en www.lexmed.de.automático, el médico
ingresa los resultados de su examen en el formulario de entrada de la figura 7.6. Después de
uno o dos segundos, recibe las probabilidades de los cuatro diagnósticos diferentes, así como
una sugerencia de tratamiento (Sección 7.3.5). Si faltan ciertos resultados del examen como
entrada (por ejemplo, los resultados del sonograma), entonces el médico elige la entrada no
examinada. Naturalmente, la certeza del diagnóstico es mayor cuando se ingresan más
valores de síntomas.

8Se puede acceder a una versión con funcionalidad limitada sin contraseña.

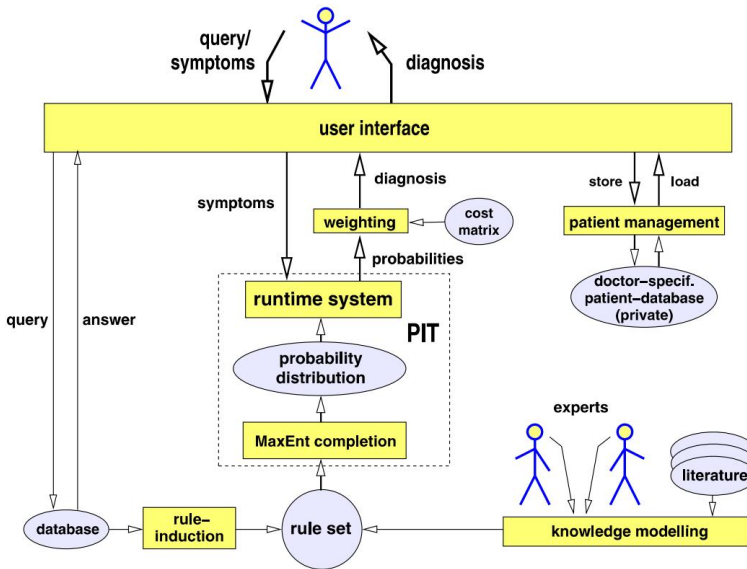


Fig. 7.7 Las reglas se generan a partir de la base de datos y del conocimiento experto. A partir de estos, MaxEnt crea una distribución de probabilidad completa. Para una consulta de usuario, se calcula la probabilidad de todos los diagnósticos posibles. Usando la matriz de costos (ver Secc. 7.3.5) se sugiere una decisión

Cada usuario registrado tiene acceso a una base de datos de pacientes privada, en la que se pueden archivar los datos de entrada. Así, los datos y diagnósticos de pacientes anteriores pueden compararse fácilmente con los de un paciente nuevo.

7.3.4 Función de LEXMED

El conocimiento se formaliza mediante proposiciones probabilísticas. Por ejemplo, la proposición

$$P(\text{Leuko7} > 20000 | \text{Diag4} = \text{inflamado}) = 0,09$$

da una frecuencia del 9% para un valor de leucocitos de más de 20.000 en caso de un apéndice inflamado⁹.

Aprendizaje de reglas por inducción estadística Los datos

sin procesar en la base de datos de LEXMED contienen 54 valores diferentes (anonizados) para 14.646 pacientes. Como se mencionó anteriormente, solo los pacientes cuyos apéndices fueron extirpados quirúrgicamente se incluyen en esta base de datos. De los 54 atributos utilizados en la base de datos, después de un análisis estadístico los 14 síntomas que se muestran en la Tabla 7.2 en la página 133

⁹En lugar de valores numéricos individuales, aquí también se pueden usar intervalos (por ejemplo [0.06, 0.12]).

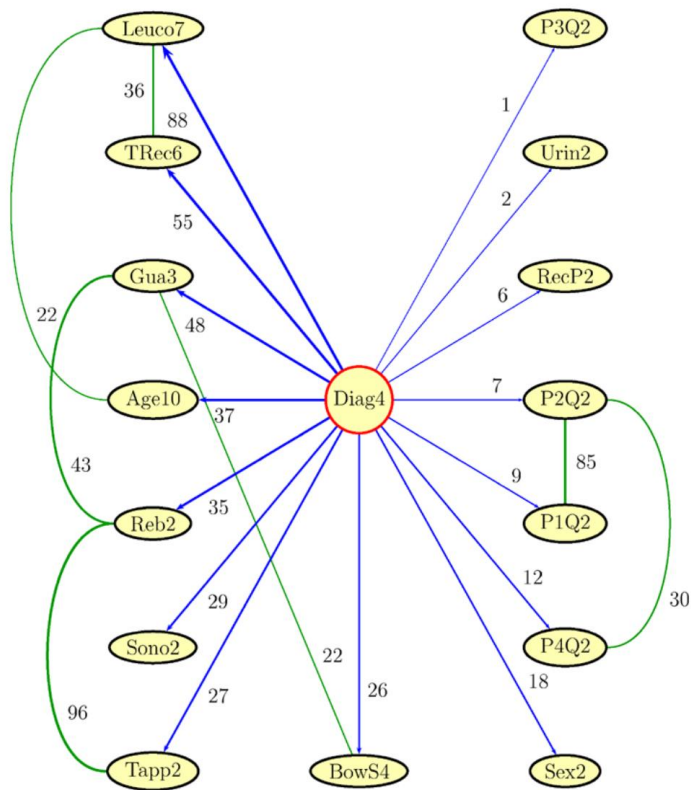


Fig. 7.8 Gráfico de dependencia calculado a partir de la base de datos

fueron usados. Ahora las reglas se crean a partir de esta base de datos en dos pasos. El primer paso determina la estructura de dependencia de los síntomas. El segundo paso llena esta estructura con las respectivas reglas de probabilidad.¹⁰

Determinación del gráfico de dependencia El gráfico de la figura 7.8 contiene para cada variable (el síntoma y el diagnóstico) un nodo y aristas dirigidas que conectan varios nodos. El grosor de los bordes entre las variables representa una medida de la dependencia estadística o correlación de las variables. La correlación de dos variables independientes es igual a cero. Se calculó la correlación de pares para cada uno de los 14 síntomas con Diag4 y se enumeró en el gráfico. Además, se calcularon todas las correlaciones triples entre el diagnóstico y dos síntomas. De estos, solo los valores más fuertes se han dibujado como bordes adicionales entre los dos síntomas participantes.

¹⁰Para una introducción sistemática al aprendizaje automático, remitimos al lector al Cap. 8.

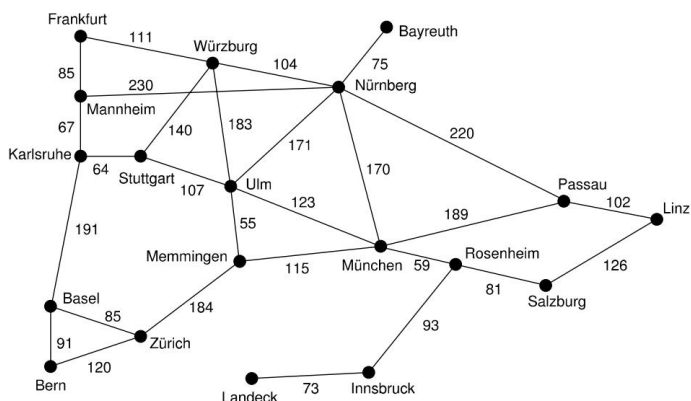


Fig. 6.5 El gráfico del sur de Alemania como ejemplo de una tarea de búsqueda con una función de coste

para b porque un árbol con factor de ramificación constante y profundidad d tiene un total de

$$\text{norte} = \frac{d}{\text{segundo} - 1} = \frac{\text{segundo re} + 1 - 1}{\text{segundo} - 1} \quad (6.2)$$

nodos.

Para la aplicación práctica de algoritmos de búsqueda para árboles de búsqueda finitos, el último nivel es especialmente importante porque

Teorema 6.1 Para árboles de búsqueda finitos muy ramificados con un factor de ramificación constante grande, casi todos los nodos están en el último nivel.

La demostración simple de este teorema se recomienda al lector como ejercicio (Ejercicio 6.1 de la página 110).

Ejemplo 6.2 Nos dan un mapa, como el representado en la Fig. 6.5, como un gráfico con ciudades como nodos y conexiones de carreteras entre las ciudades como bordes ponderados con distancias. Estamos buscando una ruta óptima de la ciudad A a la ciudad B. La descripción del esquema correspondiente dice Estado: Una ciudad como la ubicación actual del viajero.

Estado inicial: una ciudad arbitraria.

Estado objetivo: una ciudad arbitraria.

Acciones: Viajar desde la ciudad actual a una ciudad vecina.

Función de costo: La distancia entre las ciudades. Cada acción corresponde a un borde.

en el gráfico con la distancia como el peso.

Espacio de estados: Todas las ciudades, es decir, nodos del grafo.

Para encontrar la ruta con la longitud mínima, se deben tener en cuenta los costos porque no son constantes como lo eran en el 8-puzzle.

Definición 6.3 Se dice que un algoritmo de búsqueda es óptimo si, si existe una solución, siempre encuentra la solución con el costo más bajo.

El problema de los 8 acertijos es determinista, lo que significa que cada acción lleva de un estado a un único estado sucesor. Además es observable, es decir, el agente siempre sabe en qué estado se encuentra. En la planificación de rutas en aplicaciones reales no siempre se dan ambas características. La acción "Conducir de Múnich a Ulm" puede conducir, por ejemplo, a causa de un accidente, al estado sucesor "Munich". También puede ocurrir que el viajero ya no sepa dónde está porque se perdió. Queremos ignorar este tipo de complicaciones al principio. Por lo tanto, en este capítulo solo veremos problemas que son deterministas y observables.

Problemas como el rompecabezas de 8, que son deterministas y observables, hacen que la planificación de acciones sea relativamente simple porque, al tener un modelo abstracto, es posible encontrar secuencias de acción para la solución del problema sin llevar a cabo las acciones en el mundo real. En el caso del rompecabezas de 8, no es necesario mover los cuadrados en el mundo real para encontrar la solución. Podemos encontrar soluciones óptimas con los llamados algoritmos fuera de línea. Uno enfrenta desafíos muy diferentes cuando, por ejemplo, construye robots que se supone que juegan al fútbol. Aquí nunca habrá un modelo abstracto exacto de las acciones. Por ejemplo, un robot que patea la pelota en una dirección específica no puede predecir con certeza hacia dónde se moverá la pelota porque, entre otras cosas, no sabe si un oponente atrapará o desviará la pelota. Aquí se necesitan algoritmos en línea, que toman decisiones basadas en señales de sensores en cada situación. Aprendizaje por refuerzo, descrito en la Secc. 10, trabaja hacia la optimización de estas decisiones basadas en la experiencia.

6.2 Búsqueda no informada

6.2.1 Búsqueda en amplitud

En la búsqueda primero en amplitud, el árbol de búsqueda se explora de arriba a abajo de acuerdo con el algoritmo proporcionado en la Fig. 6.6 en la página 90 hasta que se encuentra una solución. En primer lugar, se prueba cada nodo de la lista de nodos para determinar si es un nodo objetivo y, en caso de éxito, se detiene el programa. De lo contrario, se generan todos los sucesores del nodo. A continuación, la búsqueda continúa recursivamente en la lista de todos los nodos recién generados. Todo se repite hasta que no se generan más sucesores.

Este algoritmo es genérico. Es decir, funciona para aplicaciones arbitrarias si se proporcionan las dos funciones específicas de la aplicación "Objetivo alcanzado" y "Sucesores". "ObjetivoAlcanzado" calcula si el argumento es un nodo objetivo y "Sucesores" calcula la lista de todos los nodos sucesores de su argumento. La Figura 6.7 en la página 90 muestra una instantánea de la búsqueda primero en amplitud.

```
AMPLIA-PRIMERO-BÚSQUEDA (Lista de nodos, Objetivo)
Nuevos Nodos =
Para todos los nodos   NodeList
    Si se alcanza el objetivo (nodo, objetivo)
        Return("Solución encontrada", Nodo)
    NewNodes = Agregar (NewNodes, Sucesores (Nodo))
Si NuevosNodos =
    Return(BÚSQUEDA PRIMERO-ANCHO(NewNodes, Goal))
Demás
    Retorno ("Sin solución")
```

Fig. 6.6 El algoritmo para la búsqueda primero en amplitud

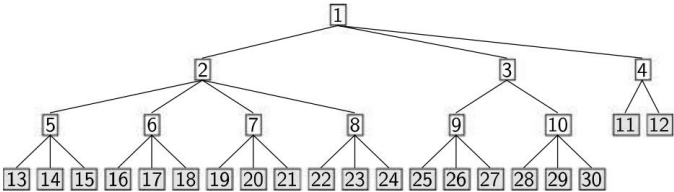


Fig. 6.7 Búsqueda en amplitud durante la expansión de los nodos de tercer nivel. Los nodos se numeran según el orden en que se generaron. Aún no se han generado los sucesores de los nodos 11 y 12

Análisis Dado que la búsqueda primero en amplitud busca completamente en todas las profundidades y alcanza todas las profundidades en un tiempo finito, es completa si el factor de ramificación b es finito. La solución óptima (es decir, la más corta) se encuentra si los costos de todas las acciones son iguales (vea el Ejercicio 6.7 en la página 110). El tiempo de cálculo y el espacio de memoria crecen exponencialmente con la profundidad del árbol. Para un árbol con un factor de ramificación b constante y una profundidad d , el tiempo total de cálculo viene dado por

$$C_{\text{total}} = \sum_{i=0}^d b^i = \frac{b^{d+1} - 1}{b - 1} = O(b^{d+1})$$

Aunque solo el último nivel se guarda en la memoria, el requisito de espacio de memoria también es $O(b^{d+1})$.

Con la velocidad de las computadoras actuales, que pueden generar miles de millones de nodos en cuestión de minutos, la memoria principal se llena rápidamente y la búsqueda finaliza. El problema de que no siempre se encuentra la solución más corta se puede resolver mediante la llamada Búsqueda de Costo Uniforme, en la que el nodo con el costo más bajo de los ordenados ascendentemente

```

PROFUNDIDAD-PRIMERO-BÚSQUEDA (Nodo, Objetivo)

Si GoalReached(Node, Goal) Return("Solución encontrada")
NewNodes = Sucesores (Nodo)
Mientras NuevosNodos =
    Resultado = PROFUNDIDAD-PRIMERO-BÚSQUEDA(Primero(NuevosNodos), Objetivo)
    Si Resultado = "Solución encontrada" Return("Solución encontrada")
    NuevosNodos = Resto(NuevosNodos)
Retorno ("Sin solución")

```

Fig. 6.8 El algoritmo para la búsqueda primero en profundidad. La función "Primero" devuelve el primer elemento de una lista, y "Resto" el resto de la lista

la lista de nodos siempre se expande y los nuevos nodos se ordenan. Así encontramos la solución óptima. Sin embargo, el problema de la memoria aún no está resuelto. Una solución para este problema es proporcionada por la búsqueda en profundidad.

6.2.2 Búsqueda en profundidad primero

En la búsqueda primero en profundidad, solo unos pocos nodos se almacenan en la memoria a la vez. Después de la expansión de un nodo, solo se guardan sus sucesores y el primer nodo sucesor se expande inmediatamente. Así, la búsqueda se vuelve rápidamente muy profunda. Solo cuando un nodo no tiene sucesores y la búsqueda falla en esa profundidad, el siguiente nodo abierto se expande retrocediendo hasta la última rama, y así sucesivamente. Podemos percibir esto mejor en el elegante algoritmo recursivo de la figura 6.8 y en el árbol de búsqueda de la figura 6.9 en la página 92.

Análisis La búsqueda primero en profundidad requiere mucha menos memoria que la búsqueda primero en amplitud porque como máximo se guardan nodos b en cada profundidad. Por lo tanto, necesitamos células de memoria $b \cdot d$.

Sin embargo, la búsqueda primero en profundidad no está completa para árboles infinitamente profundos porque la búsqueda primero en profundidad se ejecuta en un bucle infinito cuando no hay una solución en la rama más a la izquierda. Por lo tanto, la cuestión de encontrar la solución óptima es obsoleta. Debido al ciclo infinito, no se puede dar ningún límite en el tiempo de cálculo. En el caso de un árbol de búsqueda de profundidad finita con profundidad d , se generan un total de aproximadamente b nodos.

Por lo tanto, el tiempo de cálculo crece, al igual que en la búsqueda primero en amplitud, exponencialmente con la profundidad.

Podemos hacer que el árbol de búsqueda sea finito estableciendo un límite de profundidad. Ahora bien, si no se encuentra ninguna solución en el árbol de búsqueda podado, puede haber soluciones fuera del límite.

Así la búsqueda se vuelve incompleta. Hay ideas obvias, sin embargo, para completar la búsqueda.

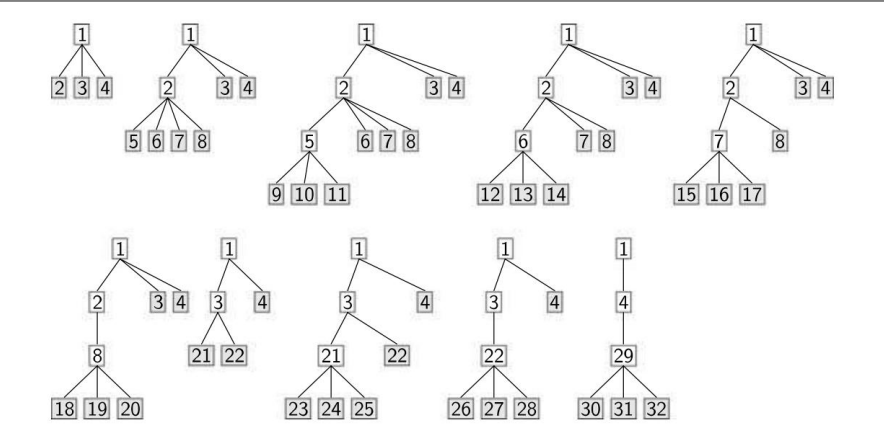


Fig. 6.9 Ejecución de búsqueda en profundidad. Todos los nodos en la profundidad tres no tienen éxito y provocan un retroceso. Los nodos están numerados en el orden en que fueron generados.

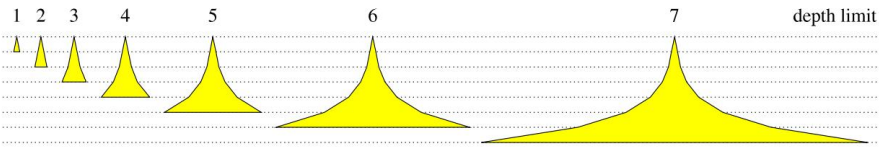


Fig. 6.10 Representación esquemática del desarrollo del árbol de búsqueda en profundización iterativa con límites de 1 a 7. La anchura del árbol corresponde a un factor de ramificación de 2

6.2.3 Profundización iterativa

Comenzamos la búsqueda primero en profundidad con un límite de profundidad de 1. Si no se encuentra una solución, elevamos el límite en 1 y comenzamos a buscar desde el principio, y así sucesivamente, como se muestra en la figura 6.10 . Este aumento iterativo del límite de profundidad se denomina profundización iterativa.

Debemos aumentar el programa de búsqueda primero en profundidad que se muestra en la Fig. 6.8 en la página 91 con los dos parámetros adicionales "Profundidad" y "Límite". "Profundidad" aumenta en uno en la llamada recursiva, y la línea principal del ciclo while se reemplaza por "While NewNodes = And Depth < Limit". El algoritmo modificado se representa en la Fig. 6.11 en la página 93.

Análisis El requisito de memoria es el mismo que en la búsqueda en profundidad. Se podría argumentar que reiniciar repetidamente la búsqueda primero en profundidad en la profundidad cero provoca una gran cantidad de trabajo redundante. Para grandes factores de ramificación, este no es el caso. Ahora mostramos que la suma del número de nodos de todas las profundidades hasta el anterior al último $d_{max} - 1$ en todos los árboles buscados es mucho menor que el número de nodos en el último árbol buscado.

PROFUNDIZACIÓN ITERATIVA(Nodo, Meta)

Límite de profundidad = 0

Repetir

Resultado = DEPTHFIRSTSEARCH-B(Nodo, Meta, 0, DepthFirstSearch)

Límite de profundidad = Límite de profundidad + 1

Hasta Resultado = "Solución encontrada"

DEPTHFIRSTSEARCH-B(Nodo, Objetivo, Profundidad, Límite)

Si GoalReached(Node, Goal) Return("Solución encontrada")

NewNodes = Sucesores (Nodo)

Mientras NuevosNodos = Y Profundidad < Límite

resultado =

DEPTHFIRSTSEARCH-B(Primero(NuevosNodos), Meta, Profundidad + 1, Límite)

Si Resultado = "Solución encontrada" Return("Solución encontrada")

NuevosNodos = Resto(NuevosNodos)

Retorno ("Sin solución")

Fig. 6.11 El algoritmo para la profundización iterativa, que llama a la búsqueda primero en profundidad ligeramente modificada con un límite de profundidad (TIEFENSUCHE-B)

Sea $N_b(d)$ el número de nodos de un árbol de búsqueda con factor de ramificación b y profundidad d y $d_{\max}$ ser la última profundidad buscada. El último árbol buscado contiene

$$N_b(d_{\max}) = \sum_{d=0}^{d_{\max}} b^d = \frac{b^{d_{\max}+1} - 1}{b - 1}$$

nodos. Todos los árboles buscados de antemano juntos tienen

$$\begin{aligned} N_b(d_{\max}) &= \sum_{d=0}^{d_{\max}} b^d = \frac{b^{d_{\max}+1} - 1}{b - 1} \\ &= \frac{1}{b - 1} \left(b^{d_{\max}+1} - 1 \right) \\ &= \frac{1}{b - 1} \left(\frac{b^{d_{\max}+1} - 1}{b - 1} \right) \\ &\approx \frac{1}{b - 1} \frac{b^{d_{\max}+1} - 1}{b - 1} = \frac{1}{b - 1} N_b(d_{\max}) \end{aligned}$$

Tabla 6.1 Comparación de los algoritmos de búsqueda no informados. (*) significa que la declaración es sólo verdadero dado un costo de acción constante. ds es la profundidad máxima para un árbol de búsqueda finito

	Amplitud primero buscar	costo uniforme buscar	Primero en profundidad buscar	Iterativo profundizando
Lo completo	Sí	Sí	No	Sí
Solucion optima	Sí (*)	Sí	No	Sí (*)
Tiempo de cómputo	re b	re b	∞ o b ds	b ^d
Uso de memoria	b ^d	b ^d	bd	bd

nodos. Para $b > 2$ esto es menor que el número $N_b(d_{max})$ de nodos en el último árbol.

Para $b = 20$, los primeros árboles $d_{max} - 1$ juntos contienen solo alrededor de $= 1/19$ de los $\frac{1}{b-1}$ número de nodos en el último árbol. El tiempo de cálculo para todas las iteraciones además de la último puede ser ignorado.

Al igual que la búsqueda en amplitud, este método es completo y tiene un costo constante para todas las acciones, encuentra la solución más corta.

6.2.4 Comparación

Los algoritmos de búsqueda descritos se han puesto uno al lado del otro en la Tabla 6.1.

Podemos ver claramente que la profundización iterativa es la ganadora de esta prueba porque Obtiene la mejor calificación en todas las categorías. De hecho, de los cuatro algoritmos presentados es el sólo uno prácticamente utilizable.

De hecho, tenemos un ganador para esta prueba, aunque para aplicaciones realistas generalmente no tiene éxito. Incluso para el rompecabezas de 15, el hermano mayor del rompecabezas de 8 (ver Ejercicio 6.4 de la página 110), hay alrededor de 2×10^{13} estados diferentes. para no trivial sistemas de inferencia el espacio de estado es muchos órdenes de magnitud más grande. Como se muestra en Secta. 6.1, toda la potencia informática del mundo no servirá de mucho más. En cambio lo que se necesita es una búsqueda inteligente que solo explore una pequeña fracción de la búsqueda espacio y encuentra una solución allí.

6.3 Búsqueda heurística

Las heurísticas son estrategias de resolución de problemas que en muchos casos encuentran una solución más rápido que la búsqueda desinformada. Sin embargo, esto no está garantizado. La búsqueda heurística podría requieren mucho más tiempo e incluso pueden resultar en que no se encuentre la solución.

Los humanos usamos con éxito procesos heurísticos para todo tipo de cosas. Cuando comprando verduras en el supermercado, por ejemplo, juzgamos las diversas opciones por una libra de fresas usando solo unos pocos criterios simples como precio, apariencia, fuente de producción, y confianza en el vendedor, y luego decidimos la mejor opción por instinto. En teoría, podría ser mejor someter las fresas a un proceso básico análisis químico antes de decidir si comprarlos. Por ejemplo, las papas de fresa pueden estar envenenadas. Si ese fuera el caso, el análisis habría valido la pena.

BÚSQUEDA HEURÍSTICA (Inicio, Meta)

Lista de nodos = [Inicio]

Si bien es cierto

Si NodeList = Return ("Sin solución")

Nodo = Primero (Lista de nodos)

Lista de nodos = Resto (Lista de nodos)

If GoalReached(Node, Goal) Return("Solución encontrada", Nodo)

NodeList = SortIn(Sucesores(Nodo),NodeList)

Fig. 6.12 El algoritmo para la búsqueda heurística

problema. Sin embargo, no llevamos a cabo este tipo de análisis porque hay una probabilidad muy alta de que nuestra selección heurística tenga éxito y nos lleve rápidamente a nuestro objetivo de comer fresas sabrosas.

Las decisiones heurísticas están íntimamente ligadas a la necesidad de tomar decisiones en tiempo real con recursos limitados. En la práctica, se prefiere una buena solución encontrada rápidamente a una solución óptima, pero muy costosa de obtener.

Se utiliza una función de evaluación heurística $f(s)$ para estados para modelar matemáticamente una heurística. El objetivo es encontrar, con poco esfuerzo, una solución al problema de búsqueda planteado con un costo total mínimo. Tenga en cuenta que existe una sutil diferencia entre el esfuerzo por encontrar una solución y el costo total de esta solución. Por ejemplo, Google Maps puede tardar medio segundo de esfuerzo en encontrar una ruta desde el ayuntamiento de San Francisco hasta Tuolumne Meadows en el Parque Nacional Yosemite, pero el viaje desde San Francisco hasta Tuolumne Meadows en coche puede tardar cuatro horas y algo de dinero. para gasolina etc. (costo total).

A continuación, modificaremos el algoritmo de búsqueda primero en anchura añadiéndole la función de evaluación. Los nodos actualmente abiertos ya no se expanden de izquierda a derecha por fila, sino de acuerdo con su clasificación heurística. Del conjunto de nodos abiertos, el nodo con la calificación mínima siempre se expande primero. Esto se logra evaluando inmediatamente los nodos a medida que se expanden y clasificándolos en la lista de nodos abiertos.

La lista puede contener nodos de diferentes profundidades en el árbol.

Debido a que la evaluación heurística de estados es muy importante para la búsqueda, a partir de ahora diferenciaremos entre estados y sus nodos asociados. La estafa del nodo contiene el estado y más información relevante para la búsqueda, como su profundidad en el árbol de búsqueda y la clasificación heurística del estado. Como resultado, la función "Sucesores", que genera los sucesores (hijos) de un nodo, también debe calcular inmediatamente para estos nodos sucesores sus calificaciones heurísticas como componente de cada nodo. Definimos el algoritmo de búsqueda general HEURISTICSEARCH en la figura 6.12.

La lista de nodos se inicializa con los nodos iniciales. Luego, en el bucle, se elimina el primer nodo de la lista y se prueba si es un nodo de solución. En caso contrario, se ampliará con la función "Sucesores" y sus sucesores se añadirán a la lista.

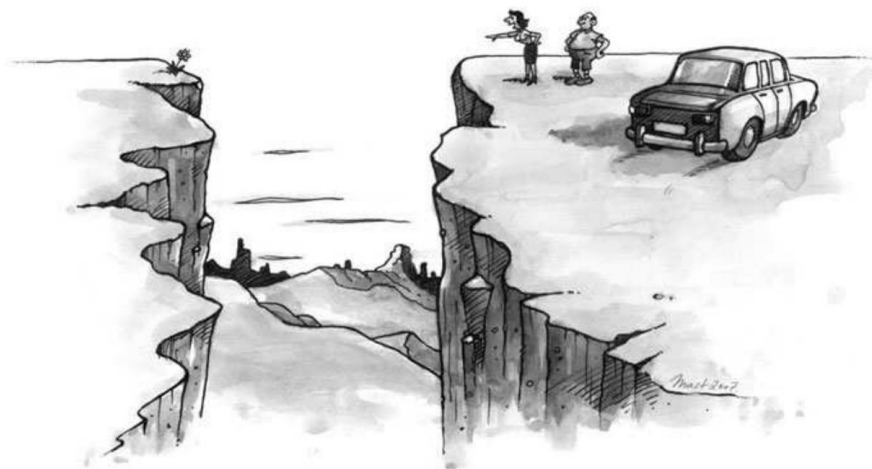


Fig. 6.13 Él: "¡Querido, piensa en el costo del combustible! Te buscaré uno en otro lugar. Ella: "¡No, yo quiero ese de allá!"

con la función "Ordenar". "SortIn(X,Y)" inserta los elementos de la lista no ordenada X en la lista ordenada ascendentemente Y. La clasificación heurística se utiliza como clave de clasificación. Por lo tanto, se garantiza que el mejor nodo (es decir, el que tiene el valor heurístico más bajo) está siempre al principio de la lista.³ La búsqueda

primero en profundidad y primero en amplitud también son casos especiales de la función BÚSQUEDA HEURÍSTICA. Podemos generarlos fácilmente conectando la función de evaluación apropiada (Ejercicio 6.11 en la página 111).

La mejor heurística sería una función que calcule los costes reales desde cada nodo hasta el objetivo. Sin embargo, hacer eso requeriría un recorrido de todo el espacio de búsqueda, que es exactamente lo que se supone que debe evitar la heurística. Por lo tanto, necesitamos una heurística que sea rápida y sencilla de calcular. ¿Cómo encontramos tal heurística?

Una idea interesante para encontrar una heurística es la simplificación del problema. La tarea original se simplifica lo suficiente como para que se pueda resolver con un costo computacional pequeño. Los costos de un estado a la meta en el problema simplificado luego sirven como una estimación para el problema real (ver Fig. 6.13). Esta función de estimación de costos la denotamos h.

6.3.1 Búsqueda codiciosa

Parece sensato elegir el estado con el valor h estimado más bajo (es decir, el que tiene el costo estimado más bajo) de la lista de estados actualmente disponibles. El

³Al ordenar un nuevo nodo de la lista de nodos, puede ser ventajoso verificar si el nodo ya está disponible y, de ser así, eliminar el duplicado.

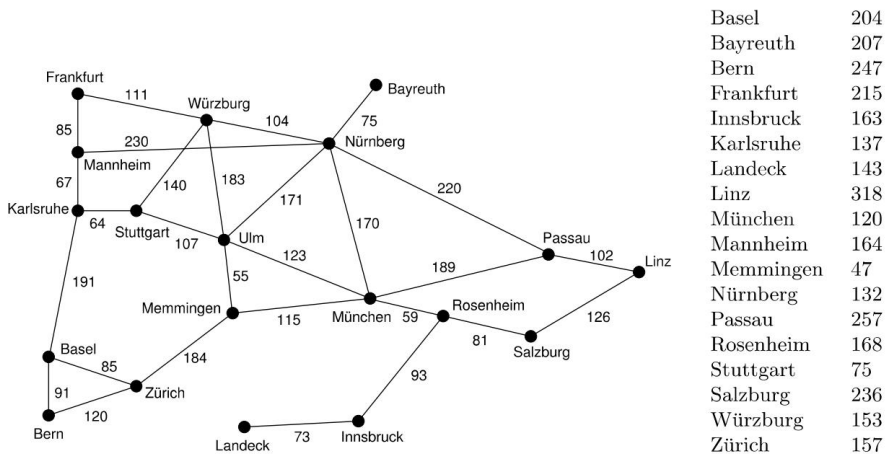


Fig. 6.14 Gráfico de ciudad con distancias de vuelo desde todas las ciudades hasta Ulm

la estimación de costes se puede utilizar directamente como función de evaluación. Para la evaluación en la función BÚSQUEDA HEURÍSTICA establecemos $f(s) = h(s)$. Esto se puede ver claramente en el ejemplo de planificación de viajes (Ejemplo 6.2 en la página 88). Planteamos la tarea de encontrar la trayectoria en línea recta de ciudad a ciudad (es decir, la distancia de vuelo) como una simplificación del problema. En lugar de buscar la ruta óptima, primero determinamos desde cada nodo una ruta con una distancia mínima de vuelo hasta el objetivo. Elegimos Ulm como destino. Por lo tanto, la función de estimación de costos se convierte en

$$h(s) = \text{distancia de vuelo desde la ciudad } s \text{ hasta Ulm.}$$

Las distancias de vuelo desde todas las ciudades hasta Ulm se muestran en la Fig. 6.14 al lado del gráfico. El árbol de búsqueda para comenzar en Linz se representa en la Fig. 6.15 en la página 98 a la izquierda. Podemos ver que el árbol es muy esbelto. La búsqueda termina así rápidamente. Desafortunadamente, esta búsqueda no siempre encuentra la solución óptima. Por ejemplo, este algoritmo no logra encontrar la solución óptima cuando comienza en Mannheim (Fig. 6.15 en la página 98 a la derecha). El camino Mannheim–Nürnberg–Ulm tiene una longitud de 401 km. La ruta Mannheim–Karlsruhe–Stuttgart–Ulm sería significativamente más corta con 238 km. A medida que observamos el gráfico, la causa de este problema se vuelve clara. De hecho, Nürnberg está algo más cerca que Karlsruhe de Ulm, pero la distancia de Mannheim a Nürnberg es significativamente mayor que la de Mannheim a Karlsruhe. La heurística solo mira "con avidez" hacia el objetivo en lugar de tener en cuenta también el tramo que ya se ha establecido hasta el nodo actual. Por eso le damos el nombre de búsqueda codiciosa.

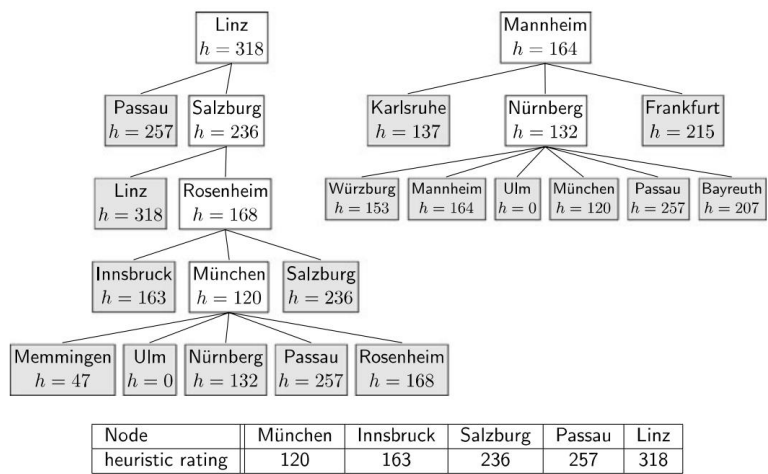


Fig. 6.15 Búsqueda codiciosa: de Linz a Ulm (izquierda) y de Mannheim a Ulm (derecha). Se proporciona la estructura de datos de la lista de nodos para el árbol de búsqueda de la izquierda, ordenados por la calificación del nodo antes de la expansión del nodo München

6.3.2 A -Buscar

Ahora queremos tener en cuenta los costes que se han acumulado durante la búsqueda hasta los nodos actuales. Primero definimos la función de costo

$g(s)$ = Suma de costos acumulados desde el inicio hasta el nodo actual,

luego agregue a eso el costo estimado para la meta y obtenga como la función de evaluación heurística

$f(s) = g(s) + h(s).$

Ahora añadimos otro requisito pequeño, pero importante.

Definición 6.4 Una función heurística de estimación de costos $h(s)$ que nunca sobrestima el costo real desde el estado s hasta la meta se llama admisible.

La función BÚSQUEDA HEURÍSTICA junto con una función de evaluación $f(s) = g(s) + h(s)$ y una función heurística admisible h se denomina algoritmo A*. Este famoso algoritmo es completo y óptimo. A* siempre encuentra la solución más corta para cada problema de búsqueda solucionable. Explicaremos y probaremos esto en la siguiente discusión.

Primero aplicamos el algoritmo A* al ejemplo. Estamos buscando el más corto camino de Frankfurt a Ulm.

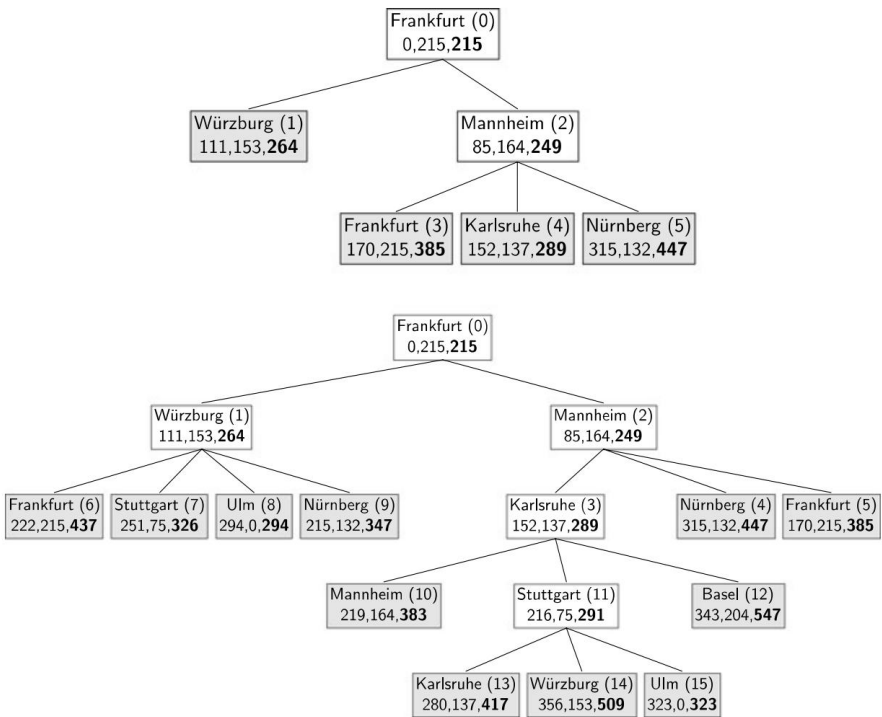


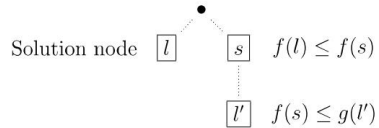
Fig. 6.16 Dos instantáneas del árbol de búsqueda A para la ruta óptima de Frankfurt a Ulm. En los recuadros debajo del nombre de la ciudad s mostramos g(s), h(s), f (s). Los números entre paréntesis después de los nombres de las ciudades muestran el orden en que los nodos han sido generados por la función "Sucesor".

En la parte superior de la Fig. 6.16 vemos que los sucesores de Mannheim se generan antes que los sucesores de Würzburg. La solución óptima Frankfurt-Würzburg-Ulm se genera poco después en el octavo paso, pero aún no se reconoce como tal. Por lo tanto, el algoritmo aún no termina porque el nodo Karlsruhe (3) tiene un valor f mejor (más bajo) y, por lo tanto, está por delante del nodo Ulm (8) en la línea. Solo cuando todos los valores de f son mayores o iguales que el del nodo de solución Ulm (8) nos aseguramos de que tenemos una solución óptima. De lo contrario, podría haber otra solución con costos más bajos. Ahora mostraremos que esto es cierto en general.

Teorema 6.2 El algoritmo A es óptimo. Es decir, siempre encuentra la solución con el menor costo total si la heurística h es admisible.

Prueba En el algoritmo BÚSQUEDA HEURÍSTICA, cada nodo s recién generado se clasifica mediante la función "SortIn" de acuerdo con su calificación heurística f (s). El nodo con el valor de calificación más bajo está al principio de la lista. Si el nodo l en el

Fig. 6.17 El primer nodo de solución l encontrado por A nunca tiene un costo mayor que otro nodo arbitrario s



principio de la lista es un nodo de solución, entonces ningún otro nodo tiene una mejor clasificación heurística. Para todos los demás nodos s es cierto entonces que $f(l) \leq f(s)$. Debido a que la heurística es admisible, no se puede encontrar una mejor solución l , incluso después de la expansión de todos los demás nodos (ver Fig. 6.17). Escrito formalmente:

$$g(l) = g(l) + h(l) = f(l) \leq f(s) = g(s) + h(s) \leq g(l').$$

La primera igualdad se cumple porque l es un nodo solución con $h(l) = 0$. La segunda es la definición de f . La tercera (des)igualdad se cumple porque la lista de nodos abiertos está ordenada en orden ascendente. La cuarta igualdad es nuevamente la definición de f . Finalmente, la última (des)igualdad es la admisibilidad de la heurística, que nunca sobrestima el costo del nodo s a una solución arbitraria. Así se ha demostrado que $g(l) \leq g(l')$, es decir, que la solución descubierta l es óptima.

6.3.3 IDA -Buscar

La búsqueda A* hereda una peculiaridad de la búsqueda en amplitud. Tiene que guardar muchos nodos en la memoria, lo que puede llevar a un uso muy alto de la memoria. Además, la lista de nodos abiertos debe estar ordenada. Por lo tanto, la inserción de nodos en la lista y la eliminación de nodos de la lista ya no pueden ejecutarse en un tiempo constante, lo que aumenta ligeramente la complejidad del algoritmo. Basándonos en el algoritmo heapsort, podemos estructurar la lista de nodos como un montón con complejidad de tiempo logarítmico para la inserción y eliminación de nodos (ver [CLR90]).

Ambos problemas se pueden resolver, de manera similar a la búsqueda primero en amplitud, mediante la profundización iterativa. Trabajamos con búsqueda en profundidad y elevamos sucesivamente el límite. Sin embargo, en lugar de trabajar con un límite de profundidad, aquí usamos un límite para la evaluación heurística $f(s)$. Este proceso se llama algoritmo IDA*.

6.3.4 Comparación empírica de los algoritmos de búsqueda

en A*, o (alternativamente) IDA*, tenemos un algoritmo de búsqueda con muchos buenos lazos de propiedad. Es completo y óptimo. Por lo tanto, se puede utilizar sin riesgo. Sin embargo, lo más importante es que funciona con heurística y, por lo tanto, puede reducir significativamente el tiempo de cálculo necesario para encontrar una solución. Nos gustaría explorar esto empíricamente en el ejemplo de 8 rompecabezas.

Para el 8-puzzle hay dos heurísticas simples admisibles. La heurística h_1 simplemente cuenta el número de cuadrados que no están en el lugar correcto. Claramente esta heurística es admisible. La heurística h_2 mide la distancia de Manhattan. Para cada

Tabla 6.2 Comparación del costo de cómputo de la búsqueda no informada y la búsqueda heurística de problemas resolubles de 8 acertijos con varias profundidades. Las medidas son en pasos y segundos. Todos los valores son promedios de varias ejecuciones (consulte la última columna)

Profundidad	Pasos iterativos de		Algoritmo A				número carreras
	profundización	Tiempo [segundo]	Heurística h1		Heurística h2		
			Pasos Tiempo [seg]		Pasos Tiempo [seg]		
2	20	0.003	3.0	0.0010	3.0	0.0010	10
4	81	0.013	5.2	0.0015	5.0	0.0022	24
6	806	0.13	10.2	0.0034	8.3	0.0039	19
8	6455	1.0	17.3	0.0060	12.2	0.0063	14
10	50512	7.9	48.1	0.018	22.1	0.011	15
12	486751	75.7	162.2	0.074	56,0	0.031	12
AIF							
14	–	–	10079.2	2.6	855.6	0.25	datos
datos	–	–	69386.6	19.0	3806.5	1.3	13
18	–	–	708780.0	161.6	53941.5	14.1	4

elevar al cuadrado las distancias horizontal y vertical hasta la ubicación de ese cuadrado en el estado objetivo se suman. Luego, este valor se suma en todos los cuadrados. por ejemplo, el Distancia de Manhattan de los dos estados



se calcula como

$$h2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$$

La admisibilidad de la distancia de Manhattan también es obvia (ver Ejercicio 6.13 sobre página 111).

Los algoritmos descritos fueron implementados en Mathematica. para una comparación con búsqueda no informada, el algoritmo A con las dos heurísticas h1 y h2 y La profundización iterativa se aplicó a 132 problemas de 8 rompecabezas generados aleatoriamente. El los valores promedio para el número de pasos y el tiempo de cálculo se dan en la Tabla 6.2. Vemos que las heurísticas reducen significativamente el costo de búsqueda en comparación con la búsqueda no informada.

Si comparamos la profundización iterativa con A con h1 en la profundidad 12, por ejemplo, se hace evidente que h1 reduce el número de pasos por un factor de aproximadamente 3,000, pero el tiempo de cálculo por sólo un factor de 1.023. Esto se debe al mayor costo por paso para el cálculo de la heurística.

Un examen más detenido revela un salto en el número de pasos entre la profundidad 12 y profundidad 14 en la columna para h1. Este salto no puede explicarse únicamente por las repetidas trabajo realizado por IDA · Surge porque la implementación del algoritmo A

elimina duplicados de nodos idénticos y, por lo tanto, reduce el espacio de búsqueda. Esto no es posible con IDA porque casi no guarda nodos. A pesar de esto, A ya no puede competir con IDA más allá de la profundidad 14 porque el costo de clasificación en nuevos nodos aumenta mucho el tiempo por paso.

Un cálculo del factor de ramificación efectivo de acuerdo con (6.1) en la página 87 arroja valores de aproximadamente 2,8 para una búsqueda no informada. Este número es consistente con el valor de la Secc. 6.1. La heurística h_1 reduce el factor de ramificación a valores de aproximadamente 1,5 y h_2 a aproximadamente 1,3. Podemos ver en la tabla que una pequeña reducción del factor de ramificación de 1,5 a 1,3 nos da una gran ventaja en el tiempo de cálculo.

Por lo tanto, la búsqueda heurística tiene un significado práctico importante porque puede resolver problemas que están fuera del alcance de una búsqueda no informada.

6.3.5 Resumen

De los diversos algoritmos de búsqueda para la búsqueda no informada, la profundización iterativa es el único práctico porque es completo y puede funcionar con muy poca memoria.

Sin embargo, para problemas difíciles de búsqueda combinatoria, incluso la profundización iterativa suele fallar debido al tamaño del espacio de búsqueda. La búsqueda heurística ayuda aquí a través de la reducción del factor de ramificación efectivo. El algoritmo IDA, como la profundización iterativa, es completo y requiere muy poca memoria.

Las heurísticas, naturalmente, solo dan una ventaja significativa si la heurística es "buena". Cuando se resuelven problemas de búsqueda difíciles, la tarea real del desarrollador consiste en diseñar heurísticas que reducen en gran medida el factor de bifurcación efectivo. Insecto. En la sección 6.5 nos ocuparemos de este problema y también mostraremos cómo se pueden usar las técnicas de aprendizaje automático para generar heurísticas automáticamente.

Para terminar, queda por notar que las heurísticas no tienen ninguna ventaja de rendimiento para los problemas irresolubles porque la irresoluble de un problema solo se puede establecer cuando se ha buscado en el árbol de búsqueda completo. Para problemas decidibles como el rompecabezas de 8, esto significa que todo el árbol de búsqueda debe recorrerse hasta una profundidad máxima, ya sea que se use una heurística o no. La heurística es siempre una desventaja en este caso, atribuible al costo computacional de evaluar la heurística. Esta desventaja generalmente se puede estimar mediante un factor constante independiente del tamaño del problema. Para problemas indecidibles como la prueba de fórmulas PL1, el árbol de búsqueda puede ser infinitamente profundo. Esto significa que, en el caso irresoluble, la búsqueda potencialmente nunca termina. En resumen, podemos decir lo siguiente: para problemas solucionables, las heurísticas suelen reducir drásticamente el tiempo de cálculo, pero para problemas irresolubles, el costo puede ser incluso mayor con heurísticas.

6.4 Juegos con oponentes

Los juegos para dos jugadores, como el ajedrez, las damas, Othello y Go, son deterministas porque cada acción (un movimiento) da como resultado el mismo estado secundario dado el mismo estado principal. Por el contrario, el backgammon no es determinista porque su estado secundario depende

en el resultado de una tirada de dados. Todos estos juegos son observables porque cada jugador siempre conoce el estado completo del juego. Muchos juegos de cartas, como el póquer, por ejemplo, son solo parcialmente observables porque el jugador no conoce las cartas de los otros jugadores, o solo tiene un conocimiento parcial sobre ellas.

Los problemas discutidos hasta ahora en este capítulo eran deterministas y observables. A continuación veremos juegos que también son deterministas y observables. Además, nos limitaremos a los juegos de suma cero. Estos son juegos en los que cada ganancia que hace un jugador significa una pérdida del mismo valor para el oponente. La suma de la ganancia y la pérdida es siempre igual a cero. Esto es cierto para los juegos de ajedrez, damas, Othello y Go, mencionados anteriormente.

6.4.1 Búsqueda Minimax

El objetivo de cada jugador es hacer movimientos óptimos que resulten en la victoria. En principio, es posible construir un árbol de búsqueda y buscar completamente a través de él (como con el rompecabezas de 8) una serie de movimientos que resultarán en la victoria. Sin embargo, hay varias peculiaridades a tener en cuenta: 1. El factor

de ramificación efectivo en el ajedrez es de alrededor de 30 a 35. En un juego típico con 50 movimientos por jugador, el árbol de búsqueda tiene más de $30^{100} \approx 10^{148}$ nodos de hojas. Por lo tanto, no hay posibilidad de explorar completamente el árbol de búsqueda. Además, el ajedrez se juega a menudo con un límite de tiempo. Debido a este requisito de tiempo real, la búsqueda debe limitarse a una profundidad adecuada en el árbol, por ejemplo, ocho medios movimientos. Dado que entre los nodos de hoja de este árbol de profundidad limitada normalmente no hay nodos de solución (es decir, nodos que terminan el juego), se utiliza una función de evaluación heurística B para las posiciones del tablero. El nivel de juego del programa depende en gran medida de la calidad de esta función de evaluación. Por lo tanto, trataremos más este tema en la Secc. 6.5.

2. A continuación llamaremos Max al jugador cuyo juego deseamos optimizar, y Min a su oponente. Los movimientos del oponente (Min) no se conocen de antemano y, por lo tanto, tampoco lo es el árbol de búsqueda real. Este problema se puede resolver elegantemente asumiendo que el oponente siempre hace el mejor movimiento que puede. Cuanto mayor sea la evaluación $B(s)$ para la posición s , mejor será la posición s para el jugador Max y peor para su oponente Min. Max intenta maximizar la evaluación de sus movimientos, mientras que Min realiza movimientos que dan como resultado una evaluación lo más baja posible.

En la Fig. 6.18 en la página 104 se proporciona un árbol de búsqueda con cuatro semimovimientos y evaluaciones de todas las hojas. La evaluación de un nodo interno se deriva recursivamente como el máximo o mínimo de sus nodos secundarios, según el nivel del nodo.

6.4.2 Poda alfa-beta

Cambiando entre maximización y minimización, podemos ahorrarnos mucho trabajo en algunas circunstancias. La poda alfa-beta funciona con la búsqueda en profundidad hasta un límite de profundidad preestablecido. De esta manera, el árbol de búsqueda se busca desde la izquierda.

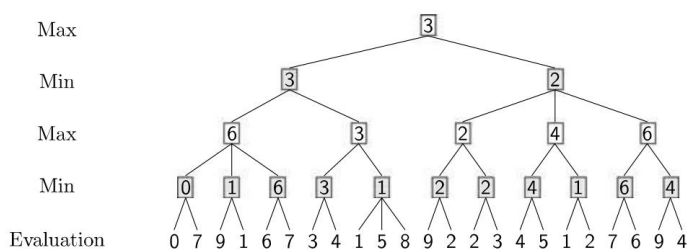


Fig. 6.18 Un árbol de juego minimax con anticipación de cuatro medios movimientos

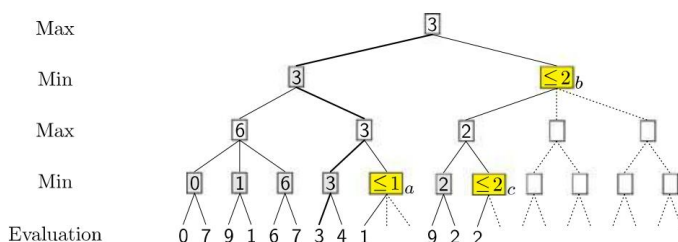


Fig. 6.19 Un árbol de juego alfa-beta con anticipación de cuatro semimovimientos. Las partes punteadas del árbol no se recorren porque no tienen efecto en el resultado final.

a derecha. Al igual que en la búsqueda minimax, en los nodos mínimos el mínimo se genera a partir del valor mínimo de los nodos sucesores y en los nodos máximos igualmente

el máximo. En la Fig. 6.19 se representa este proceso para el árbol de la Fig. 6.18. En el nodo marcado con a, todos los demás sucesores se pueden ignorar después de que el primer hijo se evalúe como el valor 1 porque el mínimo seguramente será ≤ 1 . Incluso podría volverse más pequeño aún, pero eso es irrelevante ya que el máximo ya es ≥ 3 un nivel por encima. Independientemente de cómo resulte la evaluación de los sucesores restantes, el máximo mantendrá el valor 3. Análogamente, el árbol se recortará en el nodo b. Dado que el primer hijo de b tiene el valor 2, el mínimo que se generará para b solo puede ser menor o igual a 2. Pero el máximo en el nodo raíz ya es seguro que sea ≥ 3 .

Esto no se puede cambiar por valores ≤ 2 . Por lo tanto, los subárboles restantes de b se pueden podar.

El mismo razonamiento se aplica para el nodo c. Sin embargo, el nodo máximo relevante no es el padre directo, sino el nodo raíz. Esto se puede generalizar. • En cada nodo hoja se calcula la evaluación. • Para cada nodo máximo, el valor secundario más grande actual se guarda en α . • Para cada nodo mínimo, el valor secundario más pequeño actual se guarda en β . • Si en un nodo mínimo k el valor actual $\beta \leq \alpha$, entonces la búsqueda bajo k puede terminar.

Aquí α es el valor más grande de un nodo máximo en el camino desde la raíz hasta la k. • Si en un nodo máximo l el valor actual $\alpha \geq \beta$, entonces la búsqueda debajo de l puede terminar.

Aquí β es el valor más pequeño de un nodo mínimo en el camino desde la raíz hasta la l.

```

ALFABETAMAX(Nodo,  $\alpha$ ,  $\beta$ )
If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Sucesores (Nodo)
Mientras NuevosNodos =
     $\alpha$  = Máximo( $\alpha$ , ALFABETAMIN(Primero(NuevosNodos),  $\alpha$ ,  $\beta$ ))
    Si  $\alpha \geq \beta$  Retorno( $\beta$ )
    NuevosNodos = Resto(NuevosNodos)
Retorno( $\alpha$ )

```

```

ALFABETAMIN(Nodo,  $\alpha$ ,  $\beta$ )
If DepthLimitReached(Node) Return(Rating(Node))
NewNodes = Sucesores (Nodo)
Mientras NuevosNodos =
     $\beta$  = Mínimo( $\beta$ , ALFABETAMAX(Primero(NuevosNodos),  $\alpha$ ,  $\beta$ ))
    Si  $\beta \leq \alpha$  Devuelve( $\alpha$ )
    NuevosNodos = Resto(NuevosNodos)
Retorno( $\beta$ )

```

Fig. 6.20 El algoritmo para la búsqueda alfa-beta con las dos funciones ALFABETAMIN y ALFABETAMAX

El algoritmo de la figura 6.20 es una extensión de la búsqueda en profundidad con dos funciones que se llaman alternativamente. Utiliza los valores definidos anteriormente para α y β .

Complejidad El tiempo de cálculo ahorrado por la poda alfa-beta depende en gran medida del orden en que se recorren los nodos secundarios. En el peor de los casos, la poda alfa-beta no ofrece ninguna ventaja. Para un factor de ramificación b constante, el número n_d de nodos de hoja para evaluar en la profundidad d es igual a

$$n_d = b^d.$$

En el mejor de los casos, cuando los sucesores de los nodos máximos se ordenan descendientemente y los sucesores de los nodos mínimos se ordenan ascendientemente, el factor de ramificación efectivo se reduce a $\sqrt{b}$. En ajedrez esto significa una reducción sustancial del factor de ramificación efectivo de 35 a aproximadamente 6. Entonces sólo

$$n = \sqrt[b]{n_d} = b^{d/2}$$

se crearían nodos hoja. Esto significa que el límite de profundidad y, por lo tanto, también el horizonte de búsqueda se duplican con la poda alfa-beta. Sin embargo, esto solo es cierto en el caso de sucesores ordenados de manera óptima porque las calificaciones de los nodos secundarios se desconocen en el momento en que se crean. Si los nodos secundarios se ordenan aleatoriamente, el factor de ramificación se reduce a $b^{3/4}$ y el número de nodos hoja a

$$sd = b^{\frac{3}{4} \log b}.$$

Con el mismo poder de cómputo, una computadora de ajedrez que usa la poda alfa-beta puede, por ejemplo, calcular ocho medios movimientos hacia adelante en lugar de seis, con un factor de ramificación efectivo de aproximadamente 14. Se puede encontrar un análisis completo con una derivación de estos parámetros en [Pea84].

Para duplicar la profundidad de búsqueda como se mencionó anteriormente, necesitaríamos que los nodos secundarios estén ordenados de manera óptima, lo que no es el caso en la práctica. De lo contrario, la búsqueda sería innecesaria. Con un simple truco podemos obtener un ordenamiento de nodos relativamente bueno. Conectamos la poda alfa-beta con la profundización iterativa sobre el límite de profundidad. Así en cada nuevo límite de profundidad podemos acceder a las valoraciones de todos los nodos de los niveles anteriores y ordenar los sucesores en cada rama. De este modo alcanzamos un factor de ramificación efectivo de aproximadamente 7 a 8, que no está lejos del óptimo teórico de $\sqrt{35}$ [Nil98].

6.4.3 Juegos no deterministas

La búsqueda Minimax se puede generalizar a todos los juegos con acciones no deterministas, como el backgammon. Cada jugador tira antes de su movimiento, que está influenciado por el resultado de la tirada de dados. En el árbol del juego ahora hay tres tipos de niveles en la secuencia

Max, dados, Min, dados, . . . ,

donde cada nodo de lanzamiento de dados se ramifica de seis maneras. Debido a que no podemos predecir el valor del dado, promediamos los valores de todos los lanzamientos y realizamos la búsqueda como se describe con los valores promedio de [RN10].

6.5 Funciones de evaluación heurística

¿Cómo encontramos una buena función de evaluación heurística para la tarea de búsqueda?

Aquí hay fundamentalmente dos enfoques. La forma clásica utiliza el conocimiento de expertos humanos.

Al ingeniero del conocimiento se le asigna la tarea generalmente difícil de formalizar el conocimiento implícito del experto en forma de un programa de computadora.

Ahora queremos mostrar cómo se puede simplificar este proceso en el ejemplo del programa de ajedrez.

En el primer paso, se pregunta a los expertos sobre los factores más importantes en la selección de un movimiento. Luego se intenta cuantificar estos factores. Obtenemos una lista de características o atributos relevantes. Estos son luego (en el caso más simple) combinados en

una función de evaluación lineal $B(s)$ para posiciones, que podría tener el siguiente aspecto:

$$B(s) = a_1 \cdot \text{material} + a_2 \cdot \text{peón\_estructura} + a_3 \cdot \text{rey\_seguridad} \\ + a_4 \cdot \text{caballo\_en\_el\_centro} + a_5 \cdot \text{cobertura\_diagonal\_alfil} + \dots, \quad (6.3)$$

donde "material" es, con mucho, la característica más importante y se calcula mediante

$$\text{material} = \text{material}(\text{equipo\_propio}) - \text{material}(\text{oponente})$$

con

$$\begin{aligned} \text{material}(\text{equipo}) = & \text{num\_peones}(\text{equipo}) \cdot 100 + \text{num\_knights}(\text{equipo}) \cdot 300 \\ & + \text{num\_obispos}(\text{equipo}) \cdot 300 + \text{num\_torres}(\text{equipo}) \cdot 500 \\ & + \text{num\_reinas}(\text{equipo}) \cdot 900 \end{aligned}$$

Casi todos los programas de ajedrez hacen una evaluación similar del material. Sin embargo, existen grandes diferencias para todas las demás funciones, que no abordaremos aquí [Fra05, Lar00].

En el siguiente paso se deben determinar los pesos a_i de todas las características. Estos se configuran intuitivamente después de una discusión con expertos, luego se cambian después de cada juego en función de la experiencia positiva y negativa. El hecho de que este proceso de optimización sea muy costoso y, además, que la combinación lineal de características sea muy limitada sugiere el uso de aprendizaje automático.

6.5.1 Aprendizaje de heurísticas

Ahora queremos optimizar automáticamente los pesos a_i de la función de evaluación $B(s)$ de (6.3). En este enfoque, al experto solo se le pregunta sobre las características relevantes $f_1(s), \dots, f_n(s)$ para el estado del juego s . Luego, se utiliza un proceso de aprendizaje automático con el objetivo de encontrar una función de evaluación que sea lo más cercana posible a la óptima. Comenzamos con una función de evaluación preestablecida inicial (determinada por el proceso de aprendizaje), y luego dejamos que el programa de ajedrez juegue. Al final del juego, se deriva una calificación del resultado (victoria, derrota o empate). Con base en esta calificación, la función de evaluación se cambia con el objetivo de cometer menos errores la próxima vez. En principio, el proceso de aprendizaje se ocupa automáticamente de lo mismo que hace el desarrollador.

Tan fácil como suena, es muy difícil en la práctica. Un problema central con la mejora de la calificación de la posición en función de los partidos ganados o perdidos se conoce hoy como el problema de asignación de créditos. De hecho, tenemos una calificación al final del juego, pero no calificaciones para los movimientos individuales. Así, el agente lleva a cabo muchas acciones pero no recibe ningún feedback positivo o negativo hasta el final. ¿Cómo debería entonces asignar esta retroalimentación a las muchas acciones realizadas en el pasado? ¿Y cómo debería mejorar sus acciones en ese caso? El apasionante y joven campo del aprendizaje por refuerzo se ocupa de tales cuestiones (véase la sección 10).

La mayoría de las mejores computadoras de ajedrez del mundo todavía funcionan sin técnicas de aprendizaje automático. Hay dos razones para esto. Por un lado, los algoritmos de aprendizaje por refuerzo desarrollados hasta ahora requieren una gran cantidad de tiempo de cálculo dados los grandes espacios de estado. Por otro lado, las heurísticas creadas manualmente de las computadoras de ajedrez de alto rendimiento ya están muy optimizadas. Esto significa que solo un muy buen sistema de aprendizaje puede conducir a mejoras. En los próximos diez años, presumiblemente llegará el momento en que una computadora de aprendizaje se convierta en campeona mundial.

6.6 Estado del arte

Para la evaluación de la calidad de los procesos de búsqueda heurística, me gustaría repetir Definición de Elaine Rich [Ric83]:

La inteligencia artificial es el estudio de cómo hacer que las computadoras hagan cosas en las que, en este momento, las personas son mejores.

Difícilmente existe una prueba más adecuada para decidir si un programa de computadora es inteligente como la comparación directa de una computadora y un ser humano en un juego como el ajedrez, las damas, el backgammon o el Go.

En 1950, Claude Shannon, Konrad Zuse y John von Neumann introdujeron los primeros programas de ajedrez que, sin embargo, no pudieron implementarse o llevaría mucho tiempo implementarlos. Solo unos años más tarde, en 1955, Arthur Samuel escribió un programa que jugaba a las damas y podía mejorar sus propios parámetros a través de un proceso de aprendizaje simple. Para hacer esto, utilizó la primera computadora de lógica programable, la IBM 701. Sin embargo, en comparación con las computadoras de ajedrez de hoy, tenía acceso a una gran cantidad de juegos archivados, para los cuales cada movimiento individual había sido calificado por expertos. Así, el programa mejoró su función de evaluación. Para lograr más mejoras, Samuel hizo que su programa jugara contra sí mismo. Resolvió el problema de la asignación de créditos de manera sencilla. Para cada posición individual durante un juego, compara la evaluación de la función $B(s)$ con la calculada por la poda alfa beta y cambia $B(s)$ en consecuencia. En 1961, su programa de damas venció al cuarto mejor jugador de damas de los Estados Unidos. Con este trabajo innovador, Samuel seguramente se adelantó casi 30 años a su tiempo.

Solo a principios de los años noventa, cuando surgió el aprendizaje por refuerzo, Gerald Tesauro construyó un programa de backgammon de aprendizaje llamado TD-Gammon, que jugaba al nivel de campeón mundial (ver Secc. 10).

Hoy existen varios programas de ajedrez que se juegan a nivel de gran maestro, y algunos se venden comercialmente para PC. El gran avance se produjo en 1997, cuando Deep Blue de IBM derrotó al campeón mundial de ajedrez Gary Kasparov con una puntuación de 3,5 juegos a 2,5. Deep Blue podía, en promedio, calcular 12 medios movimientos por delante con poda alfa-beta y evaluación de posición heurística.

Una de las computadoras de ajedrez más poderosas hasta la fecha es Hydra, una computadora paralela propiedad de una empresa en los Emiratos Árabes Unidos. El software fue desarrollado por los científicos Christian Donn timer (Austria) y Ulf Lorenz (Alemania), así como por el gran campeón de ajedrez alemán Christopher Lutz. Hydra usa 64 Xeon paralelos

Procesadores con aproximadamente 3 GHz de potencia informática y 1 GB de memoria cada uno. Para la función de evaluación de posición, cada procesador tiene un coprocesador FPGA (matriz de puertas programables en campo). De este modo, es posible evaluar 200 millones de posiciones por segundo incluso con una función de evaluación costosa.

Con esta tecnología, Hydra puede, en promedio, calcular alrededor de 18 movimientos por delante. En situaciones especiales y críticas, el horizonte de búsqueda puede extenderse incluso a 40 semimovimientos. Claramente, este tipo de horizonte está más allá de lo que pueden hacer incluso los grandes campeones, ya que Hydra a menudo hace movimientos que los grandes campeones no pueden comprender, pero que al final conducen a la victoria. En 2005, Hydra derrotó al séptimo gran maestro Michael Adams con 5,5 a 0,5 juegos.

Hydra utiliza poco conocimiento especial de los libros de texto sobre ajedrez, más bien búsqueda alfa-beta con heurística relativamente general y bien conocida y una buena evaluación de posición codificada a mano. En particular, Hydra no es capaz de aprender. Los desarrolladores llevan a cabo mejoras entre juegos. Por lo tanto, la computadora todavía está aliviada de tener que aprender. Hydra tampoco tiene algoritmos de planificación especiales. El hecho de que Hydra funcione sin aprender es una pista de que, a pesar de los muchos avances, todavía hay una necesidad de investigación en aprendizaje automático. Como se mencionó anteriormente, entraremos en más detalles sobre esto en la Secc. 10 y cap. 8.

En 2009, el sistema Pocket Fritz 4, que se ejecuta en una PDA, ganó el torneo de ajedrez Copa Mercosur en Buenos Aires con nueve victorias y un empate contra 10 excelentes ajedrecistas humanos, tres de ellos grandes maestros. Aunque no hay mucha información disponible sobre la estructura interna del software, esta máquina de ajedrez representa una tendencia que se aleja del poder de cómputo en bruto hacia una mayor inteligencia. Esta máquina juega a nivel de gran maestro y es comparable, si no mejor, que Hydra.

Según el desarrollador de Pocket Fritz, Stanislav Tsukrov [Wik10], Pocket Fritz con su motor de búsqueda de ajedrez HIARCS 13 busca menos de 20 000 posiciones por segundo, que es más lento que Hydra por un factor de aproximadamente 10 000. Esto lleva a la conclusión de que HIARCS 13 definitivamente usa mejores heurísticas para disminuir el factor de ramificación efectivo que Hydra y, por lo tanto, bien puede llamarse más inteligente que Hydra. Por cierto, HIARCS es una abreviatura de Sistema de ajedrez de respuesta automática de inteligencia superior.

Incluso si pronto ningún humano tendrá una oportunidad contra las mejores computadoras de ajedrez, todavía hay muchos desafíos para la IA. Ir, por ejemplo. En este antiguo juego japonés que se juega en un tablero cuadrado con 361 cuadrados, 181 piedras blancas y 180 negras, el factor de ramificación efectivo es de alrededor de 300. Después de cuatro medias jugadas ya hay alrededor de 8×10^9 posiciones. Todos los procesos de búsqueda de árboles de juegos clásicos conocidos no tienen ninguna posibilidad contra buenos jugadores humanos de Go en este nivel de complejidad. Los expertos coinciden en que aquí se necesitan sistemas "verdaderamente inteligentes". La enumeración combinatoria de todas las posibilidades es el método incorrecto. Lo que se necesita mucho más son procesos que puedan reconocer patrones en el tablero, seguir desarrollos a largo plazo y tomar decisiones "intuitivas" rápidamente. Al igual que en el reconocimiento de objetos en imágenes complejas, los humanos todavía están muy por delante de los programas informáticos. Procesamos la imagen como un todo de una manera muy paralela, mientras que la computadora procesa los millones de píxeles uno tras otro y tiene dificultades para ver lo que es esencial en la plenitud de los píxeles. El programa "Las muchas caras del Go" puede reconocer 1.100 patrones diferentes y conoce 200 estrategias de juego. Pero todos los programas de Go todavía tienen serios problemas para reconocer si un grupo de piedras está viva o muerta, o dónde agruparlas en el medio.

6.7 Ejercicios

Ejercicio 6.1 (a)

Demuestre el teorema 6.1 en la página 88, en otras palabras, demuestre que para un árbol con un gran factor de ramificación constante b , casi todos los nodos están en el último nivel a la profundidad d . (b) Demuestre que esto no siempre es cierto cuando el factor de ramificación efectivo es grande y no constante.

Ejercicio 6.2 (a)

Calcule el factor de ramificación promedio para el rompecabezas de 8 sin verificación de ciclos. El factor de ramificación promedio es el factor de ramificación que tendría un árbol con igual número de nodos en el último nivel, factor de ramificación constante e igual profundidad. (b) Calcule el factor de ramificación promedio para el rompecabezas de 8 para búsqueda no informada mientras se evitan ciclos de longitud 2.

Ejercicio 6.3 (a)

¿Cuál es la diferencia entre el factor de ramificación promedio y el efectivo?

(Definición 6.2 en la página 87)?

(b) ¿Por qué el factor de ramificación efectivo es más adecuado para el análisis y la comparación del tiempo de cálculo de los algoritmos de búsqueda que el factor de ramificación promedio? (c) Muestre que para un árbol muy ramificado con n nodos y profundidad d , el factor de ramificación efectivo b^* es aproximadamente igual al factor de ramificación promedio y, por lo tanto, igual a $\sqrt[d]{n}$.

Ejercicio 6.4 (a)

Calcule el tamaño del espacio de estado para el rompecabezas de 8, para el rompecabezas de 3 análogos (matriz de 2×2), así como para el rompecabezas de 15 (matriz de 4×4). (b)

Demuestre que el gráfico de estado que consiste en los estados (nodos) y las acciones (aristas) para el rompecabezas de 3 se divide en dos subgráficos conectados, entre los cuales no hay conexiones.

Ejercicio 6.5 Con la búsqueda primero en anchura para el rompecabezas de 8, encuentre un camino (manualmente) desde

el nodo inicial

1		3
4	2	6
7	5	8

al nodo objetivo

1	2	3
4	5	6
7	8	

Ejercicio 6.6

(a) Programe la búsqueda primero en amplitud, la búsqueda primero en profundidad y la profundización iterativa en el idioma de su elección y pruébelos en el ejemplo de 8 rompecabezas. (b) ¿Por qué tiene poco sentido usar la búsqueda primero en profundidad en el rompecabezas de 8?

Ejercicio 6.7 (a)

Muestre que la búsqueda primero en amplitud dado un costo constante para todas las acciones está garantizada para encontrar la solución más corta.

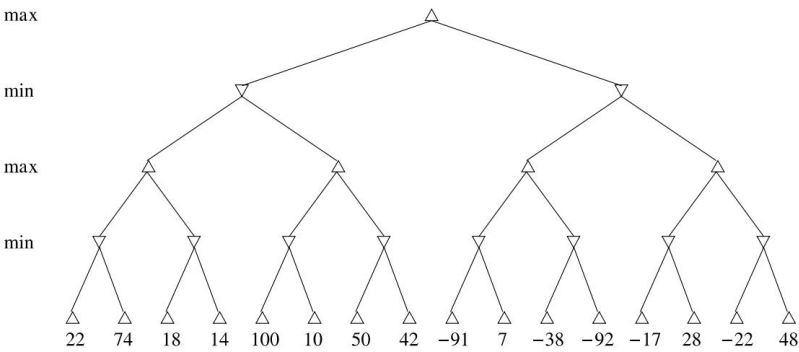


Fig. 6.21 Árbol de búsqueda Minimax

(b) Demuestre que este no es el caso para costos variables.

Ejercicio 6.8 Usando A para buscar el rompecabezas de 8, busque (manualmente) un camino desde

el nodo inicial

1		3
4	2	6
7	5	8

 al nodo objetivo

1	2	3
4	5	6
7	8	

(a) usando la heurística h1 (Sección 6.3.4). (b) utilizando la heurística h2 (Sección 6.3.4).

Ejercicio 6.9 Construya el árbol de búsqueda A para el gráfico de la ciudad de la Fig. 6.14 en la página 97 y use la distancia de vuelo a Ulm como heurística. Comience en Berna con Ulm como destino. Tenga cuidado de que cada ciudad solo aparezca una vez por camino.

Ejercicio 6.10 Programa A busca en el lenguaje de programación de tu elección utilizando las heurísticas h1 y h2 y pruébelas en el ejemplo de 8 rompecabezas.

Ejercicio 6.11 Proporcione una función de evaluación heurística para estados con los que HEURIS TICSEARCH se puede implementar como una búsqueda primero en profundidad y otra para una implementación de búsqueda primero en amplitud.

Ejercicio 6.12 ¿Cuál es la relación entre la imagen de la pareja en el cañón de la Fig. 6.13 en la página 96 y las heurísticas admisibles?

Ejercicio 6.13 Muestre que las heurísticas h1 y h2 para el rompecabezas de 8 de la Secc. 6.3.4 son admisibles.

Ejercicio 6.14 (a)

El árbol de búsqueda para un juego de dos jugadores se muestra en la figura 6.21 con las calificaciones de todos los nodos hoja. Utilice la búsqueda minimax con poda α - β de izquierda a derecha. Tacha todos los nodos que no se visitan y otorga la calificación resultante óptima para cada nodo interno. Marca el camino elegido. (b)

Ponte a prueba usando el applet de P. Winston [Win].

Para resolver este problema, se han desarrollado varias formas de lógica no monótona, que permiten eliminar conocimientos (fórmulas) de la base de conocimientos.

Bajo el nombre de lógica predeterminada, se han desarrollado lógicas que permiten asignar atributos a los objetos que son válidos mientras no haya otras reglas disponibles. En el ejemplo de Piolín, la regla que los pájaros pueden volar sería una regla predeterminada. A pesar de un gran esfuerzo, estas lógicas en la actualidad, debido a problemas semánticos y prácticos, no han tenido éxito.

La monotonía puede ser especialmente inconveniente en problemas de planificación complejos en los que el mundo puede cambiar. Si, por ejemplo, una casa azul se pinta de rojo, luego es roja. Una base de conocimiento como

```
color(casa, azul)
pintura(casa, rojo)
pintura(x, y)    color(x, y)
```

lleva a la conclusión de que, después de pintar, la casa es roja y azul. El problema que surge aquí en la planificación se conoce como el problema del marco. Una solución para esto es el cálculo de situación presentado en la Secc. 5.6.

Un enfoque interesante para modelar problemas como el ejemplo de Piolín es la teoría de la probabilidad. La afirmación "todas las aves pueden volar" es falsa. Una declaración como "casi todas las aves pueden volar" es correcta. Esta afirmación se vuelve más exacta si damos una probabilidad de que "los pájaros pueden volar". Esto lleva a la lógica probabilística, que hoy representa una subárea importante de la IA y una herramienta importante para modelar la incertidumbre (ver el Capítulo 7).

4.4 Modelado de la incertidumbre

La lógica de dos valores solo puede y debe modelar circunstancias en las que hay verdadero, falso y ningún otro valor de verdad. Por lo tanto, para muchas tareas del razonamiento cotidiano, la lógica de dos valores no es lo suficientemente expresiva. La regla

```
pájaro(x)    mosca(x)
```

es cierto para casi todas las aves, pero para algunas es falso. Como ya se mencionó, trabajar con probabilidades permite la formulación exacta de la incertidumbre. La afirmación "99% de todas las aves pueden volar" se puede formalizar con la expresión

$$P(\text{pájaro}(x) \rightarrow \text{mosca}(x)) = 0,99.$$

En el cap. 7 veremos que aquí es mejor trabajar con probabilidades condicionales como

$$P(\text{mosca} | \text{pájaro}) = 0,99.$$

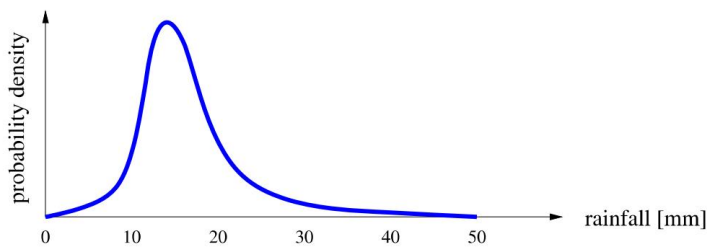


Fig. 4.3 Densidad de probabilidad de la variable continua lluvia

Con la ayuda de las redes bayesianas, las aplicaciones complejas con muchas variables pueden también ser modelado.

Se necesita un modelo diferente para la afirmación “Hace buen tiempo”. Aquí a menudo No tiene sentido hablar en términos de verdadero y falso. La variable tiempo_es_agradable no debe modelarse como binario, sino de forma continua con valores, por ejemplo, en el intervalo [0, 1]. weather_is_nice = 0.7 entonces significa "El clima es bastante agradable". La lógica difusa, que también será presentada en el Cap. 7, fue desarrollado para este tipo de variable continua (borrosa).

La teoría de la probabilidad también ofrece la posibilidad de hacer afirmaciones sobre la probabilidad de variables continuas. Una declaración en el informe meteorológico como “Hay una alta probabilidad de que llueva” podría, por ejemplo, formularse exactamente como una densidad de probabilidad de la forma

$$P(\text{lluvia} = X) = Y$$

y representado gráficamente algo como en la Fig. 4.3.

Esta representación muy general e incluso visualizable de los dos tipos de incertidumbre que hemos comentado, junto con la estadística inductiva y la teoría de las redes bayesianas, hace posible, en principio, responder a arbitrariedades probabilísticas. consultas

La teoría de la probabilidad, así como la lógica difusa, no son directamente comparables con los predicados. lógica porque no admiten variables ni cuantificadores. Pueden verse así como extensiones de la lógica proposicional como se muestra en la siguiente tabla.

Formalismo	Número de valores de verdad	probabilidades posible de expresar
Lógica proposicional	2	–
Lógica difusa	∞	–
Lógica probabilística discreta	–	Sí
Lógica probabilística continua	∞	Sí

4.5 Ejercicios

Ejercicio 4.1

- (a) Con el siguiente argumento (falso), se podría afirmar que PL1 es decidible: tomamos un cálculo de prueba completo para PL1. Con él podemos encontrar una prueba para cualquier fórmula verdadera en tiempo finito. Para cualquier otra fórmula φ procedo de la siguiente manera: aplico el cálculo a $\neg\varphi$ y demuestro que $\neg\varphi$ es verdadera. Por lo tanto, φ es falso. Así puedo probar o refutar cada fórmula en PL1. Encuentra el error en el argumento y cámbialo para que sea correcto. (b) Construya un proceso de decisión para el conjunto de fórmulas verdaderas e insatisfactorias en PL1.

Ejercicio 4.2

- (a) Dada la afirmación "Hay un barbero que afeita a toda persona que no se afeita a sí mismo". Considere si este barbero se afeita a sí mismo. (b) Sea $M = \{x|x / x\}$. Describa este conjunto y considere si M se contiene a sí mismo.

Ejercicio 4.3 Use un probador de teoremas automatizado (por ejemplo, E [Sch02]) y aplíquelo a las cinco axiomatizaciones diferentes del ejemplo de Piolín de la sección. 4.3. Valide las declaraciones del ejemplo.

Programación lógica con PROLOG

5

En comparación con los lenguajes de programación clásicos como C o Pascal, Logic lo hace posible expresar relaciones de manera elegante, compacta y declarativa. automatizado probadores de teoremas son incluso capaces de decidir si una base de conocimiento lógicamente implica una consulta. El cálculo de prueba y el conocimiento almacenado en la base de conocimiento son estrictamente separados. Una fórmula descrita en la forma normal de la cláusula se puede utilizar como entrada datos para cualquier probador de teoremas, independientemente del cálculo de prueba utilizado. esto es de gran valor para el razonamiento y la representación del conocimiento.

Si uno desea implementar algoritmos, que inevitablemente tienen componentes procedimentales, una descripción puramente declarativa a menudo es insuficiente. Roberto Kowalski, uno de los pioneros de la programación lógica, hizo este punto con la fórmula

Algoritmo = Lógica + Control.

Esta idea se llevó a cabo en el lenguaje PROLOG. PROLOG se utiliza en muchos proyectos, principalmente en IA y lingüística computacional. ahora daremos un breve introducción a este lenguaje, presentar los conceptos más importantes, mostrar su puntos fuertes y compararlo con otros lenguajes de programación y demostradores de teoremas. Aquellos que buscan un curso de programación completo se dirigen a libros de texto como [Bra11, CM94] y los manuales [Wie04, Dia04].

La sintaxis del lenguaje PROLOG solo permite cláusulas Horn. notación lógica y la sintaxis de PROLOG se yuxtaponen en la siguiente tabla:

PL1 / cláusula forma normal PROLOG					Descripción
(¬A1	...	¬Am	B) B :- A1, ... , Am.	Regla	
(A1	...	Am)	B	B :- A1, ... , Am.	Regla
A			A.		Hecho
(¬A1	...	¬Soy)	?- A1, ... , Am.		Consulta
¬(A1	...	Soy)	?- A1, ... , Am.		Consulta

```
1 hijo(oscar,karen,frank). 2
niño(Mary,Karen,Frank). 3
hijo(eva,anne,oscar). 4
hijo(Henry,Anne,Oscar). 5
hijo(isolde,anne,oscar). 6
hijo(clyde,mary,oscarb). 7 8 hijo(X,Z,Y) :-
    hijo(X,Y,Z). 9

10 descendiente(X,Y) :- hijo(X,Y,Z). 11 descendiente(X,Y) :-
    hijo(X,U,V), descendiente(U,Y).
```

Fig. 5.1 Programa PROLOG con relaciones familiares

Aquí $A_1, \dots, A_m, A, B$ son literales. Los literales, como en PL1, se construyen a partir de símbolos predicados con términos como argumentos. Como podemos ver en la tabla anterior, en PROLOG no hay negaciones en sentido lógico estricto porque el signo de un literal está determinado por su posición en la cláusula.

5.1 Sistemas e implementaciones PROLOG

Una descripción general de los sistemas PROLOG actuales está disponible en la colección de enlaces en la página de inicio de este libro. Recomendamos al lector los sistemas GNU-PROLOG [Dia04] y SWI-PROLOG, muy potentes y de libre acceso (bajo licencias públicas GNU). Para los siguientes ejemplos, se utilizó SWI-PROLOG [Wie04].

La mayoría de los sistemas PROLOG modernos funcionan con un intérprete basado en la máquina abstracta de Warren (WAM). El código fuente de PROLOG se compila en el llamado código WAM, que luego es interpretado por WAM. Las implementaciones más rápidas de un WAM gestionan hasta 10 millones de inferencias lógicas por segundo (LIPS) en una PC de 1 GHz.

5.2 Ejemplos simples

Comenzamos con las relaciones familiares del ejemplo 3.2 en la página 35. La pequeña base de conocimiento KB está codificada, sin los hechos para el predicado femenino, como un programa PROLOG llamado rel.pl en la figura 5.1.

El programa se puede cargar y compilar en el intérprete PROLOG con el comando

?- [rel].

Una consulta inicial devuelve el diálogo.

?- niño(eva, oscar, ana).

Sí

con la respuesta correcta Sí. ¿Cómo surge esta respuesta? Para la consulta "?- child(eve,oscar,anne)". hay seis hechos y una regla con el mismo predicado en su cabeza de cláusula. Ahora se intenta la unificación entre la consulta y cada uno de los literales complementarios en los datos de entrada en orden de aparición. Si una de las alternativas falla, se retrocede hasta el último punto de bifurcación y se prueba la siguiente alternativa. Debido a que la unificación falla con todos los hechos, la consulta se unifica con la regla recursiva en la línea 8. Ahora el sistema intenta resolver el subobjetivo child(eve,anne,oscar), que tiene éxito con la tercera alternativa. La consulta

?- descendiente(X,Y).

X = Óscar

y = karen

Sí

se contesta con la primera solución encontrada, como es

?- descendiente(clyde,Y).

Y = María

Sí

La consulta

?- descendiente(clyde,karen).

sin embargo, no se responde. La razón de esto es la cláusula de la línea 8, que especifica la simetría del predicado hijo. Esta cláusula se autodenomina recursivamente sin posibilidad de terminación. Este problema se puede resolver con el siguiente programa nuevo (aquí se han omitido los hechos).

```
1 descendiente(X,Y) :- hijo(X,Y,Z). 2 descendiente(X,Y) :-  
hijo(X,Z,Y). 3 descendiente(X,Y) :- hijo(X,U,V),  
descendiente(U,Y).
```

Pero ahora la consulta

?- niño(eva, oscar, ana).

ya no se responde correctamente porque ya no se da la simetría de niño en las dos últimas variables. La solución a ambos problemas se encuentra en el programa.

```
1 child_fact(oscar,karen,franz). 2
child_fact(mary,karen,franz). 3
child_fact(eva,anne,oscar). 4
child_fact(henry,anne,oscar). 5
child_fact(isolde,anne,oscar). 6
child_fact(clyde,mary,oscarb). 7 8 hijo(X,Z,Y) :-
hecho_hijo(X,Y,Z). 9 hijo(X,Z,Y) :- hecho_hijo(X,Z,Y). 10
11 descendiente(X,Y) :- hijo(X,Y,Z). 12 descendiente(X,Y) :-
hijo(X,U,V), descendiente(U,Y).
```

Al introducir el nuevo predicado `child_fact` para los hechos, el predicado `child` ya no es recursivo. Sin embargo, el programa ya no es tan elegante y simple como la primera variante —lógicamente correcta— de la figura 5.1 en la página 68, que conduce al ciclo infinito. El programador de PROLOG debe, al igual que en otros lenguajes, prestar atención al procesamiento y evitar bucles infinitos. PROLOG es solo un lenguaje de programación y no un probador de teoremas.

Debemos distinguir aquí entre semántica declarativa y procedimental de los programas PROLOG. La semántica declarativa viene dada por la interpretación lógica de las cláusulas horn. La semántica procedimental, en cambio, está definida por la ejecución del programa PROLOG, que ahora deseamos observar con más detalle. La ejecución del programa de la Fig. 5.1 en la página 68 con la consulta `child(eve,oscar,anne)` se representa en la Fig. 5.2 en la página 71 como un árbol de búsqueda.¹ La ejecución comienza en la parte superior izquierda con la consulta. Cada borde representa un posible paso de resolución SLD con un literal unificable complementario. Mientras que el árbol de búsqueda se vuelve infinitamente profundo por la regla recursiva, la ejecución de PROLOG termina porque los hechos ocurren antes que la regla en los datos de entrada.

Con la consulta `descendant(clyde,karen)`, por el contrario, la ejecución de PROLOG no termina. Podemos ver esto claramente en el árbol and-or presentado en la Fig. 5.3 en la página 71. En esta representación, las ramas, representadas por, conducen desde el , encabezado de una cláusula hasta los subobjetivos. Debido a que todos los subobjetivos de una cláusula deben resolverse, estos son y ramas. Todas las demás ramas son o ramas, de las cuales al menos una debe ser unificable con sus nodos principales. Los dos hechos reseñados representan la

¹Las constantes se han abreviado para ahorrar espacio.

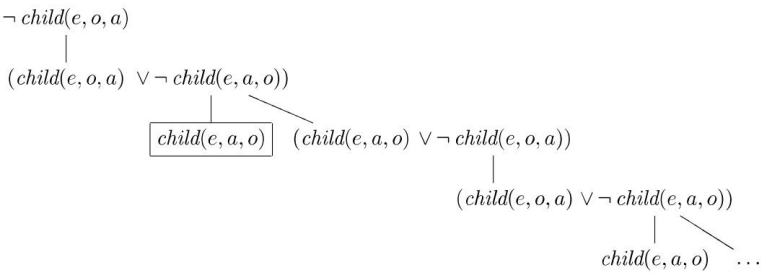


Fig. 5.2 Árbol de búsqueda de PROLOG para child(eve,oscar,anne)

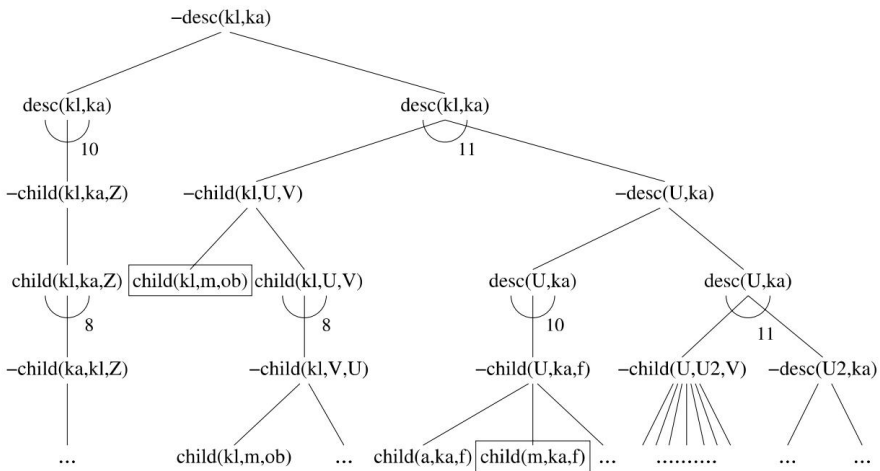


Fig. 5.3 Árbol Y-O para desc(clyde,karen)

solución a la consulta. Sin embargo, el intérprete de PROLOG no termina aquí, ya que funciona utilizando una búsqueda en profundidad con retroceso (consulte la Sección 6.2.2) y, por lo tanto, elige primero el camino infinitamente profundo hacia la izquierda.

5.3 Control de Ejecución y Elementos Procesales

Como hemos visto en el ejemplo de la relación familiar, es importante controlar la ejecución de PROLOG. Evitar retroceder innecesariamente puede conducir especialmente a grandes aumentos en la eficiencia. Un medio para este fin es el corte. Al insertar un signo de exclamación en una cláusula, podemos evitar retroceder sobre este punto. En el siguiente programa, el predicado max(X,Y,Max) calcula el máximo de los dos números X e Y.

```
1 máx(X,Y,X) :- X >= Y. 2 máx(X,Y,Y) :-
X < Y.
```

Si se aplica el primer caso (primera cláusula), entonces no se alcanzará el segundo. Por otro lado, si el primer caso no se aplica, la condición del segundo caso es verdadera, lo que significa que no es necesario verificarla. Por ejemplo, en la consulta

```
?-máx(3,2,Z), Z > 10.
```

se emplea el retroceso porque $Z=3$, y la segunda cláusula se prueba para max, que está condenada al fracaso. Por lo tanto, retroceder sobre este punto es innecesario. Podemos optimizar esto con un corte:

```
1 máx(X,Y,X) :- X >= Y, !. 2 máx. (X,Y,Y).
```

Así, la segunda cláusula solo se llama si es realmente necesaria, es decir, si la primera cláusula falla. Sin embargo, esta optimización hace que el programa sea más difícil de entender.

Otra posibilidad para el control de la ejecución es el error de predicado incorporado, que nunca es cierto. En el ejemplo de la relación familiar, podemos simplemente imprimir todos los niños y sus padres con la consulta

```
?- child_fact(X,Y,Z), write(X), write(' es un hijo de '), write(Y), write(' y '), write(Z), write('.') , nl,
fallo.
```

La salida correspondiente es

```
oscar es hijo de karen y frank. mary es hija de karen y frank.
eve es hija de anne y oscar.
```

```
...
```

```
No.
```

donde el predicado nl provoca un salto de línea en la salida. ¿Cuál sería la salida al final sin el uso del predicado fallido?

Con la misma base de conocimientos, la consulta `"?- child_fact(ulla,X,Y)."` daría como resultado la respuesta No porque no hay hechos sobre ulla. Esta respuesta no es lógicamente correcta. En concreto, no es posible probar que no existe ningún objeto con el nombre ulla. Aquí, el probador E respondería correctamente "No se encontró ninguna prueba". Por lo tanto, si PROLOG responde No, esto solo significa que la consulta Q no puede

ser probado. Para esto, sin embargo, $\neg Q$ no necesariamente debe probarse. Este comportamiento se denomina negación como fracaso.

Limitarnos a las cláusulas Horn no supone un gran problema en la mayoría de los casos. Sin embargo, es importante para la ejecución de procedimientos utilizando resolución SLD (Sección 2.5). A través del literal positivo determinado individualmente por cláusula, la resolución SLD y, por lo tanto, la ejecución de programas PROLOG, tienen un punto de entrada único en la cláusula. Esta es la única forma en que es posible tener una ejecución reproducible de programas lógicos y, por lo tanto, una semántica de procedimiento bien definida.

De hecho, ciertamente hay declaraciones de problemas que no pueden ser descritas por las cláusulas de Horn. Un ejemplo es la paradoja de Russell del Ejemplo 3.7 en la página 45, que contiene la cláusula que no es de Horn (afeitados(barbero,X) afeitados(X, X)).

5.4 Listas

Como lenguaje de alto nivel, PROLOG tiene, al igual que el lenguaje LISP, el conveniente tipo de datos de lista genérica. Una lista con los elementos A, 2, 2, B, 3, 4, 5 tiene la forma

[A,2,2,B,3,4,5]

La construcción [Head|Tail] separa el primer elemento (Head) del resto (Tail) de la lista. Con la base de conocimientos

lista([A,2,2,B,3,4,5]).

PROLOG muestra el diálogo

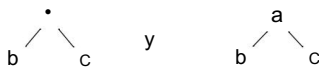
?- lista([H|T]).

H=A

T = [2, 2, segundo, 3, 4, 5]

Sí

Mediante el uso de listas anidadas, podemos crear estructuras de árbol arbitrarias. Por ejemplo, los dos árboles



puede ser representado por las listas [b,c] y [a,b,c], respectivamente, y los dos árboles



por las listas `[[e,f,g],[h],d]` y `[a,[b,e,f,g],[c,h],d]`, respectivamente.
En los árboles donde los nodos internos contienen símbolos, el símbolo es la cabeza del list y los nodos secundarios son la cola.

Un buen y elegante ejemplo de procesamiento de listas es la definición del predicado `append(X,Y,Z)` para agregar la lista Y a la lista X. El resultado se guarda en Z. El programa PROLOG correspondiente lee

```
1 añadir ([],L,L).
2 añadir ([X|L1],L2,[X|L3]) :- añadir(L1,L2,L3).
```

Esta es una descripción lógica declarativa (recursiva) del hecho de que L3 resulta de agregando L2 a L1. Al mismo tiempo, sin embargo, este programa también hace el trabajo cuando se llama. La llamada

?- añadir ([a,b,c],[d,1,2],Z).

devuelve la sustitución `Z = [a, b, c, d, 1, 2]`, al igual que la llamada

?- añadir(X,[1,2,3],[4,5,6,1,2,3]).

produce la sustitución `X = [4, 5, 6]`. Aquí observamos que `append` no es un función de dos lugares, sino una relación de tres lugares. En realidad, también podemos introducir el "parámetro de salida" Z y pregunte si se puede crear.

La inversión del orden de los elementos de una lista también se puede describir con elegancia y si programado simultáneamente por el predicado recursivo

```
1 nrev([],[]).
2 nrev([H|T],R) :- nrev(T,RT), agregar(RT,[H],R).
```

que reduce la inversión de una lista a la inversión de una lista que es un elemento menor. De hecho, este predicado es muy ineficiente debido a que llama a `append`. Este programa se conoce como ingenuo inverso y se utiliza a menudo como punto de referencia PROLOG (ver Ejercicio 5.6 de la página 81). Las cosas van mejor cuando uno procede usando un temporal almacén, conocido como el acumulador, de la siguiente manera:

Lista	Acumulador
[a B C D] []	
[b, c, d]	[a]
[cd]	(Invertido en Listas)
[d]	[c,b,a]
[]	[d,c,b,a]

El programa PROLOG correspondiente lee

```
1 acumulado([],A,A). 2
  accrev([H|T],A,R) :- accrev(T,[H|A],R).
```

5.5 Programas automodificables

Los programas PROLOG no están completamente compilados, sino que son interpretados por el WAM. Por lo tanto, es posible modificar programas en tiempo de ejecución. Un programa puede incluso modificarse a sí mismo. Con comandos como afirmar y retractarse, se pueden agregar hechos y reglas a la base de conocimientos o eliminarlos.

Una aplicación simple de la variante asserta es la adición de hechos derivados al comienzo de la base de conocimiento con el objetivo de evitar una derivación repetida, potencialmente costosa en tiempo (vea el Ejercicio 5.8 en la página 81). Si en nuestro ejemplo de relación familiar reemplazamos las dos reglas para el predicado descendiente con

```
1 :- descendiente dinámico/2. 2
  descendiente(X,Y) :- hijo(X,Y,Z), asserta(descendiente(X,Y)). 3
  descendiente(X,Y) :-
    hijo(X,U,V), descendiente(U,Y), 4 asserta(descendiente(X,Y)).
```

luego, todos los hechos derivados para este predicado se guardan en la base de conocimiento y, por lo tanto, en el futuro no se vuelven a derivar. La consulta

```
?- descendiente(clyde, karen).
```

conduce a la suma de los dos hechos

```
descendiente(clyde, karen).
```

```
descendiente(mary, karen).
```

Mediante la manipulación de reglas con aserción y retracción, incluso se pueden escribir programas que cambian por sí mismos por completo. Esta idea se conoció bajo el término programación genética. Permite la construcción de programas de aprendizaje arbitrariamente flexibles. En la práctica, sin embargo, resulta que, debido a la gran cantidad de posibles cambios sin sentido, cambiar el código por ensayo y error rara vez conduce a un aumento del rendimiento. El cambio sistemático de reglas, por otro lado, hace que la programación sea mucho más compleja que, hasta ahora, los programas que modifican extensamente su propio código no han tenido éxito. En el cap. 8 mostraremos cómo el aprendizaje automático ha tenido bastante éxito. Sin embargo, aquí solo se están realizando modificaciones muy limitadas del código del programa.

```

1 inicio: - acción (estado (izquierda, izquierda, izquierda, izquierda),
2               estado (derecha, derecha, derecha, derecha)).
3
4 acciones (Inicio, Meta): -
      plan(Inicio,Objetivo[Inicio],Ruta),
5 6 nl,escribir("Solución:."),nl,
7 write_path(Ruta).
8 % write_path(Ruta), falla. 9 % de salida de todas las soluciones

10 plan (Inicio, Meta, Visitado, Ruta): -
11 ir (Inicio, Siguiente),
12 seguro(Siguiente),
13 \+ miembro(Siguiente,Visitado), % no(miembro(...))
14     plan (Siguiente, Objetivo, [Siguiente | Visitado], Ruta).
15 plan(Objetivo,Objetivo,Camino,Camino).

17 go(estado(X,X,Z,K),estado(Y,Y,Z,K)):-across(X,Y). % granjero, lobo
18 go(estado(X,W,X,K),estado(Y,W,Y,K)):-across(X,Y). % granjero, cabra
19 go(estado(X,W,Z,X),estado(Y,W,Z,Y)):-across(X,Y). % agricultor, repollo
20 go(estado(X,W,Z,K),estado(Y,W,Z,K)):-across(X,Y). % agricultor
21
22 de ancho (izquierda, derecha).
23 de ancho (derecha, izquierda).
24
25 seguro (estado (B, W, Z, K)): - a través (W, Z), a través (Z, K).
26 seguro(estado(B,B,B,K)).
27 seguro (estado (B, W, B, B)).

```

Fig. 5.4 Programa PROLOG para el problema granjero-lobo-cabra-repollo

5.6 Un ejemplo de planificación

Ejemplo 5.1 El siguiente acertijo sirve como enunciado de un problema para un programa PRO LOG típico.

Un granjero quiere cruzar un río con un repollo, una cabra y un lobo, pero su bote es muy pequeño. que solo puede llevarlos a través de uno a la vez. El granjero lo pensó y luego le dijo a mismo: "Si primero llevo al lobo al otro lado, entonces la cabra se comerá el repollo. Si yo transporta el repollo primero, luego el lobo se comerá la cabra. ¿Qué tengo que hacer?"

Esta es una tarea de planificación que podemos resolver rápidamente con un poco de reflexión. El El programa PROLOG que se muestra en la Fig. 5.4 no se crea tan rápido.

El programa funciona en términos de estado de forma (Ganadero, Lobo, Cabra, Repollo), que describen el estado actual del mundo. Las cuatro variables con los valores posibles izquierda, derecha dan la ubicación de los objetos. El plan de predicado recursivo central primero crea un estado sucesor Luego, usando go, prueba su seguridad con seguro, y lo repite recursivamente hasta que los estados de inicio y objetivo sean los mismos (en la línea de programa 15). Los estados que ya han sido visitados se almacenan en el tercer argumento del plan. Con el miembro de predicado integrado se prueba si el state Next ya ha sido visitado. Si es así, no se intenta. La definición de la Aquí falta el predicado write_path para la tarea de generar el plan encontrado. Él se sugiere como ejercicio para el lector (Ejercicio 5.2 en la página 80). Para pro-

Gram prueba que el literal `write_path(Path)` se puede reemplazar con `write(Path)`.

Para la consulta "?- inicio". obtenemos la respuesta

Solución:

Granjero y cabra de izquierda a derecha

Agricultor de derecha a izquierda

Granjero y lobo de izquierda a derecha

Granjero y cabra de derecha a izquierda

Granjero y pollo de izquierda a derecha

Agricultor de derecha a izquierda

Granjero y cabra de izquierda a derecha

Sí

Para una mejor comprensión describimos la definición de plan en lógica:

$$z \text{ plan}(z, z) \quad s \quad z \quad n \quad [ir(s, n) \quad seguro(n) \quad plan(n, z) \quad plan(s, z)]$$

Esta definición resulta significativamente más concisa que en PROLOG. Hay dos razones para esto. Por un lado, la salida del plan descubierto no es importante para la lógica.

Además, no es realmente necesario verificar si el próximo estado ya fue visitado si los viajes innecesarios no molestan al agricultor. Sin embargo, si `!+ miembro(...)` se deja fuera del programa PROLOG, entonces hay un bucle infinito y es posible que PROLOG no encuentre un programa, incluso si lo hay. La causa de esto es la estrategia de búsqueda de encadenamiento hacia atrás de PROLOG, que, de acuerdo con el principio de búsqueda primero en profundidad (Sección 6.2.2), siempre trabaja en subobjetivos uno a la vez sin restringir la profundidad de recursión y, por lo tanto, es incompleta. Esto no le sucedería a un probador de teoremas con un cálculo completo.

Como en todas las tareas de planificación, el estado del mundo cambia a medida que se llevan a cabo las acciones de un paso al siguiente. Esto sugiere enviar el estado como variable a todos los predicados que dependen del estado del mundo, como en el predicado `seguro`. Las transiciones de estado ocurren en el predicado `go`. Este enfoque se denomina cálculo de situación [RN10]. Nos familiarizaremos con una extensión interesante para aprender secuencias de acción en mundos no deterministas parcialmente observables en el Cap. 10

5.7 Programación Lógica de Restricciones

La programación de sistemas de planificación, en los que deben cumplirse muchas (a veces complejas) condiciones lógicas y numéricas, puede resultar muy costosa y difícil con los lenguajes de programación convencionales. Aquí es precisamente donde la lógica podría ser útil. Uno simplemente escribe todas las condiciones lógicas en PL1 y luego ingresa una consulta. Por lo general, este enfoque falla miserablemente. La razón es el problema de los pingüinos discutido en la Secc. 4.3. El hecho de que `penguin(tweety)` asegure que `penguin(tweety)` sea

verdadero. Sin embargo, no descarta que el cuervo (tweety) también sea cierto. Descartar esto con axiomas adicionales es muy inconveniente (Sección 4.3).

La Programación Lógica de Restricciones (CLP), que permite la formulación explícita de restricciones para variables, ofrece un mecanismo elegante y muy eficiente para resolver este problema. El intérprete monitorea constantemente la ejecución del programa para el cumplimiento de todas sus restricciones. El programador se libera por completo de la tarea de controlar las restricciones, lo que en muchos casos puede simplificar enormemente la programación.

Esto se expresa en la siguiente cita de Eugene C. Freuder de [Fre97]:

La programación de restricciones representa uno de los enfoques más cercanos que la ciencia informática ha hecho hasta ahora al Santo Grial de la programación: el usuario establece el problema, la computadora lo resuelve.

Sin entrar en la teoría del problema de satisfacción de restricciones (CSP), aplicaremos el mecanismo CLP de GNU-PROLOG al siguiente ejemplo.

Ejemplo 5.2 La secretaria de la escuela secundaria Albert Einstein tiene que idear un plan para asignar salones para los exámenes finales. Tiene la siguiente información: los cuatro profesores Mayer, Hoover, Miller y Smith dan pruebas de las materias Alemán, Inglés, Matemáticas y Física en las salas numeradas de forma ascendente 1, 2, 3 y 4. Cada profesor da una prueba de exactamente una sujeto en exactamente una habitación. Además de eso, él sabe lo siguiente sobre los profesores y sus materias.

1. El Sr. Mayer nunca prueba en la sala 4.
2. El Sr. Miller siempre prueba alemán.
3. El Sr. Smith y el Sr. Miller no dan pruebas en habitaciones vecinas.
4. La Sra. Hoover prueba Matemáticas.
5. La física siempre se prueba en la habitación número 4.
6. En la sala 1 no se examinan alemán e inglés.

¿Quién da una prueba en qué habitación?

En la Fig. 5.5 en la página 79 se proporciona un programa GNU-PROLOG para resolver este problema. Este programa funciona con las variables Mayer, Hoover, Miller, Smith, así como con alemán, inglés, matemáticas, física, cada una de las cuales puede tomar un número entero. valor de 1 a 4 como número de habitación (líneas de programa 2 y 5). Un Mayer = 1 y German = 1 vinculantes significa que el Sr. Mayer da la prueba de alemán en la sala 1. Las líneas 3 y 6 aseguran que las cuatro variables particulares toman valores diferentes. La línea 8 asegura que a todas las variables se les asigne un valor concreto en el caso de una solución. Esta línea no es absolutamente necesaria aquí. Sin embargo, si hubiera varias soluciones, solo se generarían intervalos. En las líneas 10 a 16 se dan las restricciones, y las líneas restantes muestran los números de sala para todos los profesores y todas las materias en un formato simple.

El programa se carga en GNU-PROLOG con "[raumplan.pl]", y con "comienzo". obtenemos la salida

```
[3,1,2,4] [2,3,1,4]
```

```
1 inicio :-
2 dominio_fd([Mayer, Hoover, Miller, Smith],1,4),
3 fd_todos_diferentes([Mayer, Miller, Hoover, Smith]),
4
5 fd_domain([Alemán, Inglés, Matemáticas, Física],1,4),
6 fd_all_different([Alemán, Inglés, Matemáticas, Física]),
7
8 fd_labeling([Mayer, Hoover, Miller, Smith]),
9
10 Mayer #=4, 11 Miller #=
Alemán, 12 dist(Miller,Smith) #>= 2, 13
Hoover #= Matemáticas, 14 Física #= 4, 15 Alemán
#=1, 16 Inglés #=1 , 17 nl,

% Mayer no en la habitación 4
% Miller prueba alemán
% Distancia Miller/Smith >= 2
% Hoover prueba matemáticas
% Física en sala 4
% alemán no en la habitación 1
% inglés no en la habitación 1

18 escribe([Mayer, Hoover, Miller, Smith]), nl,
19 write([Alemán, Inglés, Matemáticas, Física]), nl.
```

Fig. 5.5 Programa CLP para el problema de programación de habitaciones

Representado algo más convenientemente, tenemos el siguiente horario de habitaciones:

Número de habitación 1	2	3	4
Maestro	hoover molinero	mayer smith	
Sujeto	Matemáticas	alemán inglés	física

GNU-PROLOG tiene, como la mayoría de los otros lenguajes CLP, un llamado dominio finito Solucionador de restricciones, con el que a las variables se les puede asignar un rango finito de enteros. No es necesario que sea un intervalo como en el ejemplo. También podemos ingresar una lista de valores Como ejercicio se invita al usuario, en el Ejercicio 5.9 de la página 81, a crear un Programa CLP, por ejemplo con GNU-PROLOG, para un rompecabezas de lógica no tan simple. Este rompecabezas, supuestamente creado por Einstein, se puede resolver muy fácilmente con un CLP sistema. Si intentáramos usar PROLOG sin restricciones, por otro lado, podríamos rechinar fácilmente los dientes. Cualquiera que encuentre una solución elegante con PROLOG o un probador, por favor deja que encuentre su camino hacia el autor.

5.8 Resumen

Unificación, listas, programación declarativa y la visión relacional de los procedimientos, en el que un argumento de un predicado puede actuar como entrada y salida, permite el desarrollo de programas cortos y elegantes para muchos problemas. Muchos programas serían significativamente más largo y, por lo tanto, más difícil de entender si está escrito en un procedimiento idioma. Además, estas características del lenguaje ahorran tiempo al programador. Por lo tanto, PROLOG también es una herramienta interesante para la creación rápida de prototipos, en particular para la IA.

aplicaciones La extensión CLP de PROLOG es útil no solo para acertijos lógicos, sino también para muchas tareas de optimización y programación.

Desde su invención en 1972, PROLOG se ha convertido en Europa en uno de los principales lenguajes de programación de Europa en IA, junto con los lenguajes de procedimiento. En los EE. UU., por otro lado, el lenguaje LISP inventado de forma nativa domina el mercado de IA.

PROLOG no es un probador de teoremas. Esto es intencional, porque un programador debe ser capaz de controlar el procesamiento de manera fácil y flexible, y no llegaría muy lejos con un probador de teoremas. Por otro lado, PROLOG no es muy útil por sí solo para probar teoremas matemáticos. Sin embargo, ciertamente hay probadores de teoremas interesantes que están programados en PROLOG.

Recomendados como literatura avanzada son [Bra11] y [CM94], así como el manuales [Wie04, Dia04] y, sobre el tema de CLP, [Bar98].

5.9 Ejercicios

Ejercicio 5.1 Intenta probar el teorema de la Secc. 3.7 sobre la igualdad de los elementos neutrales a la izquierda y a la derecha de los semigrupos con PROLOG. ¿Qué problemas surgen? ¿Cuál es la causa de esto?

Ejercicio 5.2

(a) Escriba un predicado `write_move(+State1, +State2)`, que genere una oración como "Granjero y lobo cruzan de izquierda a derecha" para cada cruce de bote. Estado1 y Estado2 son términos de la forma `estado(Granjero, Lobo, Cabra, Repollo)`. (b) Escriba un predicado recursivo

`write_path(+Path)`, que llama al predicado `write_move(+State1, +State2)` y genera todas las acciones del granjero.

Ejercicio 5.3

- (a) A primera vista, la variable `Path` en el plan de predicados del programa PROLOG del Ejemplo 5.1 en la página 76 es innecesaria porque aparentemente no se cambia en ninguna parte. ¿Para qué se necesita?
- (b) Si agregamos un error al final de la acción en el ejemplo, todas las soluciones se darán como salida. ¿Por qué cada solución ahora se imprime dos veces? ¿Cómo puedes prevenir esto?

Ejercicio 5.4

- (a) Muestre probando que el probador de teoremas E (en contraste con PROLOG), dada la base de conocimientos de la Fig. 5.1 en la página 68, responde a la consulta `"?- descendiente(clyde, karen)"`. correctamente. ¿Porqué es eso? (b)

Compare las respuestas de PROLOG y E para la consulta `"?- descendant(X, Y)"`.

Ejercicio 5.5 Escriba un programa PROLOG tan corto como sea posible que produzca 1024 unos.

Ejercicio 5.6 Investigar el comportamiento en tiempo de ejecución del predicado inverso ingenuo.

(a) Ejecute PROLOG con la opción de rastreo y observe las llamadas recursivas de `nrev`, `append` y `accrev`.

(b) Calcule la complejidad temporal asintótica de `append(L1,L2,L3)`, es decir, la dependencia del tiempo de ejecución de la longitud de la lista para listas grandes. Suponga que el acceso a la cabeza de una lista arbitraria lleva un tiempo constante. (c) Calcule la complejidad temporal de `nrev(L,R)`. (d) Calcule la complejidad temporal de `accrev(L,R)`.

(e) Determinar experimentalmente la complejidad temporal de los predicados `nrev`, `append` y `accrev`, por ejemplo, realizando mediciones de tiempo (`time(+Goal)` proporciona inferencias y tiempo de CPU).

Ejercicio 5.7 Use símbolos de función en lugar de listas para representar los árboles dados en la Secc. 5.4 en la página 73.

Ejercicio 5.8 La sucesión de Fibonacci se define recursivamente por $\text{fib}(0) = 1$, $\text{fib}(1) = 1$ y $\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$. (a) Defina un predicado PROLOG recursivo `fib(N,R)` que calcule `fib(N)` y lo devuelve en `R`.

(b) Determinar la complejidad del tiempo de ejecución del predicado `fib` teóricamente y por medición.

(c) Cambie su programa mediante el uso de `asserta` de modo que ya no se lleven a cabo inferencias innecesarias. (d) Determine

la complejidad del tiempo de ejecución del predicado modificado teóricamente y por medición (observe que esto depende de si `fib` se llamó previamente).

(e) ¿Por qué `fib` con `asserta` también es más rápido cuando se inicia por primera vez correctamente? después de iniciar PROLOG?

Ejercicio 5.9 El siguiente acertijo lógico típico fue supuestamente escrito por Albert Einstein. Además, supuestamente afirmó que solo el 2% de la población mundial es capaz de resolverlo. Se dan las siguientes afirmaciones.

- Hay cinco casas, cada una pintada de un color diferente.
- Cada casa está ocupada por una persona de diferente nacionalidad.
- Cada residente prefiere una bebida específica, fuma una marca específica de cigarrillos y tiene una mascota específica.
- Ninguna de las cinco personas bebe lo mismo, fuma lo mismo o tiene la

misma
mascota.

- Consejos: – El británico vive en la casa roja.
- El sueco tiene un perro.
- Al danés le gusta beber té.
- La casa verde está a la izquierda de la casa blanca.

- El dueño de la casa verde toma café.
- La persona que fuma Pall Mall tiene un pájaro.
- El hombre que vive en la casa del medio bebe leche.
- El dueño de la casa amarilla fuma Dunhill.
- El noruego vive en la primera casa.
- El fumador de Marlboro vive al lado del que tiene gato.
- El hombre del caballo vive al lado del que fuma Dunhill.
- Al fumador de Winfield le gusta beber cerveza.
- El noruego vive al lado de la casa azul.
- El alemán fuma Rothmanns.
- El fumador de Marlboro tiene un vecino que bebe agua.

Pregunta: ¿A quién pertenece el pez? (a) Primero

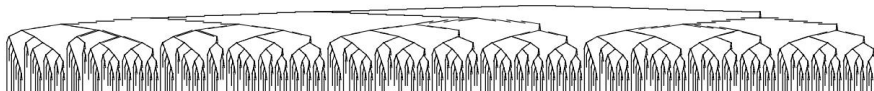
resuelve el rompecabezas manualmente.

(b) Escriba un programa CLP (por ejemplo, con GNU-PROLOG) para resolver el rompecabezas.

Orientese con el problema de programación de habitaciones en la Fig. 5.5 en la página 79.

6.1 Introducción

La búsqueda de una solución en un árbol de búsqueda extremadamente grande presenta un problema para casi todos los sistemas de inferencia. Desde el estado inicial hay muchas posibilidades para el primer paso de inferencia. Para cada una de estas posibilidades hay de nuevo muchas posibilidades en el siguiente paso, y así sucesivamente. Incluso en la prueba de una fórmula muy simple de [Ert93] con tres cláusulas Horn, cada una con tres literales como máximo, el árbol de búsqueda para la resolución SLD tiene la siguiente forma:



El árbol se cortó a una profundidad de 14 y tiene una solución en el nodo de la hoja marcado con . Solo es posible representarlo debido al pequeño factor de ramificación de dos como máximo y un corte en la profundidad 14. Para problemas realistas, el factor de ramificación y la profundidad de la primera solución pueden ser significativamente mayores.

Suponga que el factor de ramificación es una constante igual a 30 y la primera solución está a una profundidad de 50. El árbol de búsqueda tiene $30^{50} \approx 7,2 \times 10^{73}$ nodos hoja. Pero el número de pasos de inferencia es aún mayor porque no solo cada nodo hoja, sino también cada nodo interior del árbol corresponde a un paso de inferencia. Por lo tanto, debemos sumar los nodos en todos los niveles y obtener el número total de nodos del árbol de búsqueda.

$$\begin{aligned}
 50 \\
 30d = &= \frac{1 - 30^{51}}{1 - 30} \\
 re=0
 \end{aligned}$$

lo que no cambia mucho el recuento de nodos. Evidentemente, casi todos los nodos de este árbol de búsqueda están en el último nivel. Como veremos, esto es generalmente el caso. Pero ahora volvamos al árbol de búsqueda con los nodos de $7,4 \times 10^{73}$. Supongamos que tenemos 10 000 computadoras, cada una de las cuales puede realizar mil millones de inferencias por segundo, y que pudiéramos

distribuir el trabajo entre todas las computadoras sin costo alguno. El tiempo de cálculo total para todas las inferencias de $7,4 \times 10^{73}$ sería aproximadamente igual a

$$\frac{7,4 \times 10^{73} \text{ inferencias}}{\text{inferencias/seg}} = \frac{7,4 \times 10^{60} \text{ s}}{2,3 \times 10^{53} \text{ años}} \approx 2,3 \times 10^{53} \text{ años}, 10000 \times 10^9$$

que es aproximadamente 1043 veces más tiempo que la edad de nuestro universo. Mediante este simple ejercicio de pensamiento, podemos reconocer rápidamente que no existe una posibilidad realista de buscar completamente en este tipo de espacio de búsqueda con los medios disponibles para nosotros en este mundo. Además, las suposiciones relacionadas con el tamaño del espacio de búsqueda fueron completamente realistas. En el ajedrez, por ejemplo, hay más de 30 movimientos posibles para una situación típica, y un juego que dura 50 medios turnos es relativamente corto.

¿Cómo puede ser entonces que haya buenos ajedrecistas y hoy en día también buenos ordenadores de ajedrez? ¿Cómo puede ser que los matemáticos encuentren demostraciones de teoremas en los que el espacio de búsqueda es aún mucho mayor? Evidentemente los humanos utilizamos estrategias inteligentes que reducen drásticamente el espacio de búsqueda. El ajedrecista experimentado, al igual que el matemático experimentado, por la mera observación de la situación, inmediatamente descartará muchas acciones como sin sentido. A través de su experiencia, tiene la capacidad de evaluar varias acciones por su utilidad para alcanzar la meta. A menudo, una persona se guiará por el tacto. Si uno le pregunta a un matemático cómo encontró una prueba, puede responder que la intuición le llegó en un sueño. En casos difíciles, muchos médicos encuentran un diagnóstico puramente táctil, basado en todos los síntomas conocidos. Especialmente en situaciones difíciles, a menudo no existe una teoría formal para encontrar soluciones que garantice una solución óptima. En los problemas cotidianos, como la búsqueda de un gato fugitivo en la Fig. 6.1 en la página 85, la intuición juega un papel importante. Nos ocuparemos de este tipo de método de búsqueda heurística en la Secc. 6.3 y, además, describir los procesos con los que las computadoras pueden, de manera similar a los humanos, mejorar sus estrategias de búsqueda heurística mediante el aprendizaje.

Primero, sin embargo, debemos entender cómo funciona la búsqueda desinformada, es decir, probar ciegamente todas las posibilidades. Comenzamos con algunos ejemplos.

Ejemplo 6.1 Con el 8-puzzle, un ejemplo clásico de algoritmos de búsqueda [Nil98, RN10], los diversos algoritmos se pueden ilustrar muy visiblemente. Los cuadrados con los números del 1 al 8 se distribuyen en una matriz de 3×3 como la de la Fig. 6.2 en la página 86. El objetivo es llegar a un cierto orden de los cuadrados, por ejemplo en orden ascendente por filas como se representa en la Fig. 6.2 en la página 86. En cada paso, se puede mover un cuadrado hacia la izquierda, hacia la derecha, hacia arriba o hacia abajo en el espacio vacío. Por tanto, el espacio vacío se mueve en la dirección opuesta correspondiente. Para el análisis del espacio de búsqueda conviene fijarse siempre en los posibles movimientos del campo vacío.

El árbol de búsqueda para un estado inicial se representa en la Fig. 6.3 en la página 86. Podemos ver que el factor de ramificación alterna entre dos, tres y cuatro. Promediado sobre dos niveles a la vez, obtenemos un factor1 de ramificación promedio de $\sqrt[2]{8} \approx 2.83$. Vemos que cada estado se repite varias veces dos niveles más profundos porque en una simple búsqueda desinformada, cada acción se puede revertir en el siguiente paso.

1El factor de ramificación promedio de un árbol es el factor de ramificación que tendría un árbol con un factor de ramificación constante, la misma profundidad y la misma cantidad de nudos foliares.

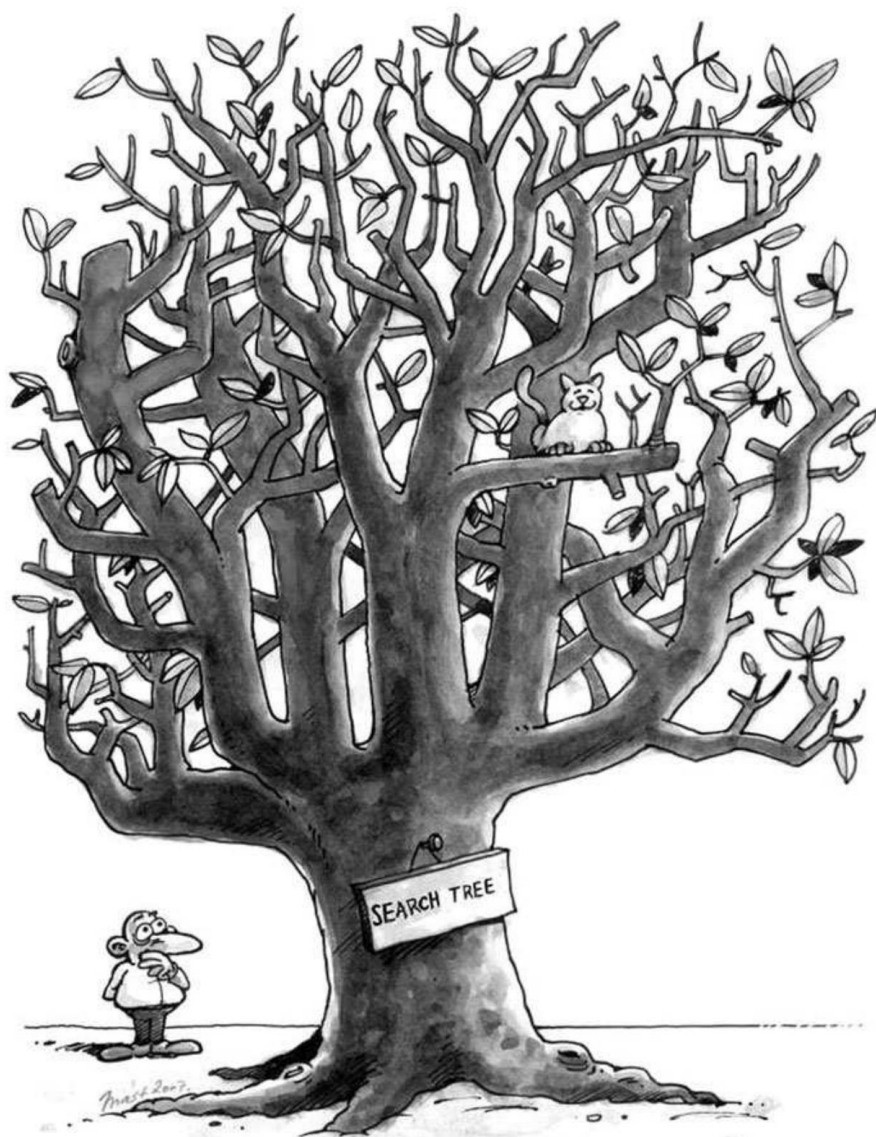


Fig. 6.1 Un árbol de búsqueda muy recortado, o: "¿Dónde está mi gato?"

Si no permitimos ciclos de longitud 2, entonces para el mismo estado inicial obtenemos el árbol de búsqueda representado en la Fig. 6.4 en la página 86. El factor de ramificación promedio se reduce en aproximadamente 1 y se convierte en 1.8.2

²Para un rompecabezas de 8, el factor de ramificación promedio depende del estado inicial (vea el Ejercicio 6.2 en página 110).

Fig. 6.2 Posibles estados inicial y final del rompecabezas de 8

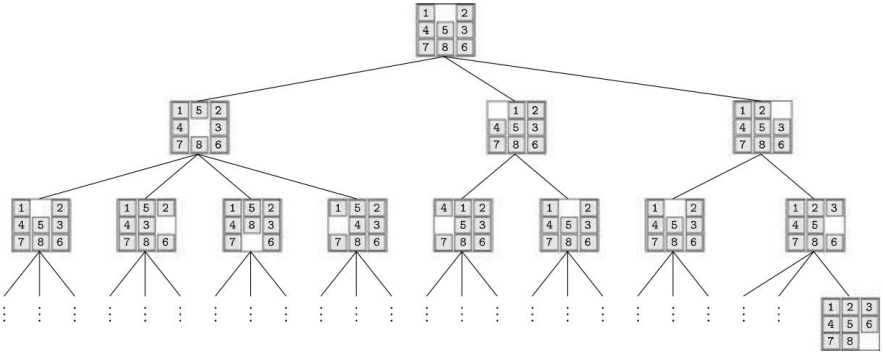


Fig. 6.3 Árbol de búsqueda para el 8-puzzle. Abajo a la derecha se representa un estado objetivo en profundidad 3. Para ahorrar espacio, se han omitido los otros nodos de este nivel.

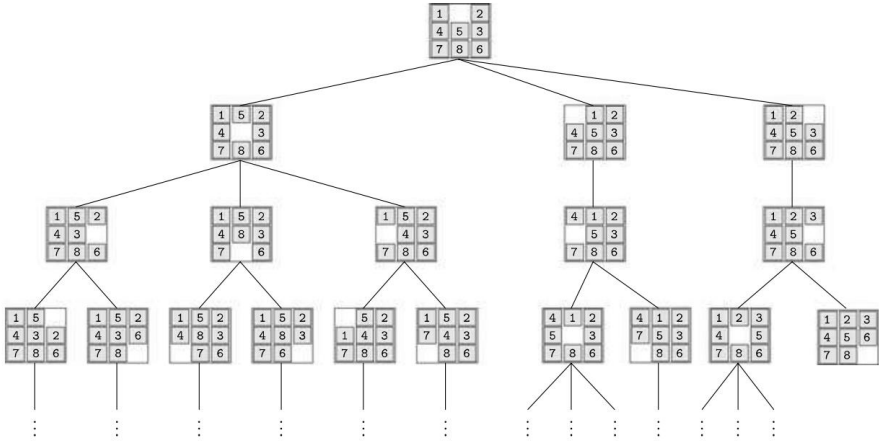


Fig. 6.4 Árbol de búsqueda para un rompecabezas de 8 sin ciclos de longitud 2

Antes de comenzar a describir los algoritmos de búsqueda, se necesitan algunos términos nuevos. Estamos lidiando con problemas de búsqueda discreta aquí. Estando en el estado s , una acción a_1 = conduce a un nuevo s' . Así $s' = a_1(s)$. Una acción diferente puede conducir al estado s'' , en otras palabras: $s'' = a_2(s)$. Aplicación recursiva de todas las acciones posibles a todos los estados, comenzando con el estado inicial, produce el árbol de búsqueda.

Definición 6.1 Un problema de búsqueda se define por los siguientes valores Estado: Descripción del estado del mundo en el que el agente de búsqueda encuentra si mismo.

Estado inicial: el estado inicial en el que se inicia el agente de búsqueda.

Estado objetivo: si el agente alcanza un estado objetivo, finaliza y genera una solución (si se desea).

Acciones: Todos los agentes permitieron acciones.

Solución: la ruta en el árbol de búsqueda desde el estado inicial hasta el estado objetivo.

Función de costo: Asigna un valor de costo a cada acción. Necesario para encontrar una solución rentable.

Espacio de estados: Conjunto de todos los estados.

Aplicado al 8-puzzle, obtenemos

Estado: matriz S de 3×3 con los valores 1, 2, 3, 4, 5, 6, 7, 8 (una vez cada uno) y uno vacío cuadrado.

Estado inicial: Un estado arbitrario.

Estado objetivo: Un estado arbitrario, por ejemplo, el estado dado a la derecha en la Fig. 6.2 en la página 86.

Acciones: Desplazamiento del cuadrado vacío Sij a la izquierda (si j = 1), derecha (si j = 3), arriba (si i = 1), abajo (si i = 3).

Función de coste: La función constante 1, ya que todas las acciones tienen igual coste.

Espacio de estados: El espacio de estados es degenerado en dominios que son mutuamente inalcanzables (Ejercicio 6.4 en la página 110). Por lo tanto, hay problemas de 8 rompecabezas irresolubles.

Para el análisis de los algoritmos de búsqueda, se necesitan los siguientes términos:

Definición 6.2 • El

número de estados sucesores de un estado s se denomina factor de ramificación b(s), o b si el factor de ramificación es constante.

- El factor de ramificación efectivo de un árbol de profundidad d con n nodos totales se define como el factor de ramificación que tendría un árbol con factor de ramificación constante, igual profundidad e igual n (ver Ejercicio 6.3 en la página 110).
- Un algoritmo de búsqueda se dice completo si encuentra una solución para cada problema solucionable. Si un algoritmo de búsqueda completo termina sin encontrar una solución, entonces el problema no tiene solución.

Para una profundidad dada d y un número de nodos n, el factor de ramificación efectivo puede ser calculado resolviendo la ecuación

$$\text{nóte} = \frac{\text{segundo } re+1 - 1}{\text{segundo} - 1} \quad (6.1)$$

Se recomienda precaución si una variable cuantificada aparece fuera del alcance de su cuantificación. tificador, como por ejemplo x en

$$\forall x (p(x) \rightarrow \exists x q(x)).$$

Aquí una de las dos variables debe ser renombrada, y en

$$\forall x (p(x) \rightarrow \exists y q(y))$$

el cuantificador se puede traer fácilmente al frente, y obtenemos como salida la fórmula equivalente

$$\forall x \exists y (p(x) \rightarrow q(y)).$$

Sin embargo, si deseamos traer correctamente el cuantificador al frente de

$$(\exists x (p(x) \rightarrow \forall y q(y))) \quad (3.4)$$

primero escribimos la fórmula en la forma equivalente

$$\neg (\exists x (p(x) \rightarrow \forall y q(y))).$$

El primer cuantificador universal ahora se convierte en

$$(\forall x \neg (p(x) \rightarrow \forall y q(y)))$$

y ahora los dos cuantificadores finalmente se pueden avanzar para

$$\forall x \exists y \neg (p(x) \rightarrow q(y)),$$

que es equivalente a

$$\forall x \exists y (p(x) \wedge \neg q(y)).$$

Vemos entonces que en (3.4) no podemos simplemente llevar ambos cuantificadores al frente. Más bien, primero debemos eliminar las implicaciones para que no haya negaciones en los cuantificadores. En general, sostiene que solo podemos extraer cuantificadores si las negaciones solo existen directamente en las subfórmulas atómicas.

Ejemplo 3.5 Como es bien sabido en análisis, la convergencia de una serie $(a_n)_{n \in \mathbb{N}}$ a un límite a está definida por

$$\varepsilon > 0 \quad n_0 \in \mathbb{N} \quad n > n_0 \Rightarrow |a_n - a| < \varepsilon.$$

Con la función $\text{abs}(x)$ para $|x|$, $a(n)$ para a_n , $\text{minus}(x, y)$ para $x - y$ y los predicados $\text{el}(x, y)$ para $x = y$, $\text{gr}(x, y)$ para $x > y$, la fórmula dice

$$\epsilon (gr(\epsilon, 0) \quad n0 (el(n0, N) \quad n (gr(n, n0) \quad gr(\epsilon, abs(menos(a(n), a)))))) \quad (3.5)$$

Esto claramente no está en la forma normal de prenex. Debido a que las variables de los cuantificadores internos $n0$ y n no aparecen a la izquierda de sus respectivos cuantificadores, no se debe cambiar el nombre de ninguna variable. A continuación eliminamos las implicaciones y obtenemos

$$\epsilon (\neg gr(\epsilon, 0) \quad n0 (\neg el(n0, N) \quad n (\neg gr(n, n0) \quad gr(\epsilon, abs(menos(a(n), a)))))).$$

Como cada negación está delante de una fórmula atómica, adelantamos los cuantificadores, eliminamos los paréntesis redundantes y con

$$\epsilon \quad n0 \quad n (\neg gr(\epsilon, 0) \quad \neg el(n0, N) \quad \neg gr(n, n0) \quad gr(\epsilon, abs(menos(a(n), a))))$$

se convierte en una cláusula cuantificada en forma conjuntiva normal.

La fórmula transformada es equivalente a la fórmula de salida. El hecho de que esto la transformación siempre es posible está garantizada por

Teorema 3.2 Toda fórmula lógica de predicados se puede transformar en una fórmula equivalente en forma normal prenex.

Además, podemos eliminar todos los cuantificadores existenciales. Sin embargo, la fórmula resultante de la llamada Skolemización ya no es equivalente a la fórmula de salida. Su satisfacibilidad, sin embargo, permanece sin cambios. En muchos casos, especialmente cuando se quiere mostrar la insatisfacibilidad de $KB \quad \neg Q$, esto es suficiente. La siguiente fórmula en forma prenex normal ahora será esquemática:

$$\forall x_1 \forall x_2 \forall y_1 \forall x_3 \forall y_2 p(f(x_1), x_2, y_1) \quad q(y_1, x_3, y_2).$$

Como la variable y_1 aparentemente depende de x_1 y x_2 , cada aparición de y_1 se reemplaza por una función de Skolem $g(x_1, x_2)$. Es importante que g sea un nuevo símbolo de función que aún no ha aparecido en la fórmula. Obtenemos

$$\forall x_1 \forall x_2 \forall x_3 \forall y_2 p(f(x_1), x_2, g(x_1, x_2)) \quad q(g(x_1, x_2), x_3, y_2)$$

y reemplace y_2 análogamente por $h(x_1, x_2, x_3)$, lo que conduce a

$$\forall x_1 \forall x_2 \forall x_3 p(f(x_1), x_2, g(x_1, x_2)) \quad q(g(x_1, x_2), x_3, h(x_1, x_2, x_3)).$$

Debido a que ahora todas las variables están universalmente cuantificadas, los cuantificadores universales pueden omitirse, lo que da como resultado

$$p(f(x_1), x_2, g(x_1, x_2)) \quad q(g(x_1, x_2), x_3, h(x_1, x_2, x_3)).$$

TRANSFORMACIÓN DE FORMA NORMAL (Fórmula): 1.

Transformación a forma normal prenex: Transformación a

forma normal conjuntiva (Teorema 2.1):

Eliminación de equivalencias.

Eliminación de implicaciones.

Aplicación repetida de la ley de Morgan y la ley distributiva.

Cambio de nombre de las variables si es necesario.

Factorización de cuantificadores universales.

2. Skolemización:

Sustitución de variables existencialmente cuantificadas por nuevas funciones de Skolem.

Supresión de los cuantificadores universales resultantes.

Fig. 3.2 Transformación de fórmulas de lógica de predicados en forma normal

Ahora podemos eliminar el cuantificador existencial (y por lo tanto también el cuantificador universal) en (3.5) en la página 39 introduciendo la función de Skolem $n_0(\epsilon)$. El prenexo esquelemitizado y la forma normal conjuntiva de (3.5) en la página 39 se lee así

$$\neg gr(\epsilon, 0) \quad \neg el(n_0(\epsilon), N) \quad \neg gr(n, n_0(\epsilon)) \quad gr(\epsilon, \text{abs}(\text{menos}(a(n), a))).$$

Al eliminar la variable n_0 , la función Skolem puede recibir el nombre n_0 .

Al skolemizar una fórmula en forma normal prenex, se eliminan todos los cuantificadores existenciales de afuera hacia adentro, donde una fórmula de la forma $x_1 \dots x_n \ y$ se reemplaza por $x_1 \dots x_n \ [y/f(x_1, \dots, x_n)]$, durante el cual f puede no aparecer en y .

Si un cuantificador existencial está en el extremo exterior, como en $\exists y p(y)$, entonces y debe reemplazarse por una constante (es decir, por un símbolo de función de lugar cero).

El procedimiento para transformar una fórmula en forma normal conjuntiva se resume en el pseudocódigo representado en la figura 3.2. La skolemización tiene un tiempo de ejecución polinomial en el número de literales. Cuando se transforma a la forma normal, la cantidad de literales en la forma normal puede crecer exponencialmente, lo que puede generar un tiempo de cálculo exponencial y un uso de memoria exponencial. La razón de esto es la aplicación repetida de la ley distributiva. El problema real, que resulta de un gran número de cláusulas, es la explosión combinatoria del espacio de búsqueda para una prueba de resolución posterior. Sin embargo, existe un algoritmo de transformación optimizado que solo genera muchos literales polinómicos [Ede91].

3.4 Cálculos de prueba

Para razonar en lógica de predicados, se han desarrollado varios cálculos de razonamiento natural como el cálculo Gentzen o el cálculo secuente. Como sugiere el nombre, estos cálculos están destinados a ser aplicados por humanos, ya que las reglas de inferencia son más

Tabla 3.2 Demostración simple con modus ponens y eliminación de cuantificadores

WB:	1	niño(eva, ana, oscar)	
WB:	2	$x \quad y \quad z$	hijo(x, y, z) hijo(x, z, y)
$E(2) : x/eva, y/anne, z/oscar$	3	niño(eva, anne, oscar)	niño(eva, oscar, ana)
MP(1, 3)	4	niño(eva, oscar, ana)	

o menos intuitivo y los cálculos funcionan con fórmulas PL1 arbitrarias. En la siguiente sección nos concentraremos principalmente en el cálculo de resolución, que es en la práctica el cálculo eficiente y automatizable más importante para fórmulas en conjuntiva normal forma. Aquí, usando el Ejemplo 3.2 en la página 35 , daremos una demostración "natural" muy pequeña. Usamos la regla de inferencia

$$\frac{A, A \quad B}{B} \quad (\text{modus ponens, MP}) \text{ y } \quad \frac{x A}{A[x/t]} \quad (\text{-eliminación, } E).$$

El modus ponens ya es familiar de la lógica proposicional. Al eliminar cuantificadores universales hay que tener en cuenta que la variable cuantificada x debe ser reemplazado por un término básico t, lo que significa un término que no contiene variables. la prueba de child(eve, oscar, anne) a partir de una base de conocimientos apropiadamente reducida en la Tabla 3.2.

Las dos fórmulas de la base de conocimientos reducida se enumeran en las filas 1 y 2. En la fila 3 se eliminan los cuantificadores universales de la fila 2, y en la fila 4 se elimina la afirmación derivado con modus ponens.

El cálculo que consta de las dos reglas de inferencia dadas no está completo. Sin embargo, puede extenderse a un procedimiento completo mediante la adición de más inferencias. normas. Este hecho no trivial es de fundamental importancia para las matemáticas y la IA. El lógico austriaco Kurt Gödel probó en 1931 que [Göd31a]

Teorema 3.3 (teorema de completitud de Gödel) La lógica de predicados de primer orden es completo. Es decir, hay un cálculo con el que toda proposición que es consecuencia de una base de conocimiento KB puede ser probada. Si $KB \models \phi$, entonces se cumple que $KB \vdash \phi$.

Toda proposición verdadera en la lógica de predicados de primer orden es, por lo tanto, demostrable. Pero es ¿lo contrario también es cierto? ¿Todo lo que podemos derivar sintácticamente es realmente cierto? El la respuesta es "sí":

Teorema 3.4 (Corrección) Hay cálculos con los que sólo se pueden demostrar proposiciones verdaderas. Es decir, si $KB \vdash \phi$ se cumple, entonces $KB \models \phi$.



Fig. 3.3 La máquina lógica universal

De hecho, casi todos los cálculos conocidos son correctos. Después de todo, tiene poco sentido trabajar con métodos de prueba incorrectos. Por lo tanto, la demostrabilidad y la consecuencia semántica son conceptos equivalentes, siempre que se utilice un cálculo correcto y completo. De este modo la lógica de predicados de primer orden se convierte en una poderosa herramienta para las matemáticas y la IA. El los cálculos de deducción natural antes mencionados son bastante inadecuados para la automatización. Sólo el cálculo de resolución, que se introdujo en 1965 y esencialmente funciona con solo una regla de inferencia simple permitió la construcción de poderosos probadores de orem automatizados, que luego se emplearon como máquinas de inferencia para sistemas expertos.

3.5 Resolución

De hecho, el cálculo de resolución correcto y completo desencadenó una euforia lógica durante la década de 1970. Muchos científicos creían que uno podía formular casi todas las tareas de representación del conocimiento y razonamiento en PL1 y luego resolverlo con un probador automatizado. La lógica de predicados, un lenguaje poderoso y expresivo, junto con un el cálculo de prueba completa parecía ser la máquina inteligente universal para representar el conocimiento y resolver muchos problemas difíciles (Fig. 3.3).

Si uno alimenta un conjunto de axiomas (es decir, una base de conocimiento) y una consulta en una máquina lógica como entrada, la máquina busca una prueba y la devuelve, porque existe una y se encontrará, como salida. Con el teorema de completitud de Gödel y el trabajo de Herbrand como base, se invirtió mucho en la mecanización de la lógica. La visión de una máquina que podría, con una base de conocimiento PL1 arbitraria y no contradictoria, probar cualquier consulta verdadera fue muy tentadora. En consecuencia, hasta ahora se están desarrollando y realizando muchos cálculos de prueba para PL1 en forma de probadores de teoremas. Como ejemplo, aquí describimos el cálculo de resolución históricamente importante y ampliamente utilizado y mostramos sus capacidades. La razón para seleccionar la resolución como ejemplo de cálculo de prueba en este libro es, como se dijo, su importancia histórica y didáctica. Hoy en día, la resolución representa solo uno de los muchos cálculos que se utilizan en probadores de alto rendimiento.

Comenzamos tratando de compilar la prueba de la tabla 3.2 en la página 41 con la base de conocimientos del ejemplo 3.2 en una prueba de resolución. Primero las fórmulas se transforman en forma normal conjuntiva y la consulta negada

$$\neg Q \equiv \neg \text{niño}(\text{eva}, \text{oscar}, \text{ana})$$

se añade a la base de conocimientos, lo que da

$$\begin{aligned} \text{KB} \quad \neg Q &\equiv (\text{niño}(\text{eva}, \text{anne}, \text{oscar}))1 \\ &(\neg \text{niño}(x, y, z) \quad \text{niño}(x, z, y))2 \\ &(\neg \text{niño}(\text{eva}, \text{oscar}, \text{ana}))3. \end{aligned}$$

La prueba podría ser algo así como

$$\begin{aligned} (2) \ x/\text{eva}, y/\text{ana}, z/\text{oscar} : &(\neg \text{niño}(\text{eva}, \text{ana}, \text{oscar}) \quad \text{niño}(\text{eva}, \\ &\text{oscar}, \text{ana}))4 \\ \text{Res}(3, 4) : &(\neg \text{niño}(\text{eva}, \text{anne}, \text{oscar}))5 \\ \text{Res}(1, 5) : &()6, \end{aligned}$$

donde, en el primer paso, las variables x,y,z se reemplazan por constantes. Luego siguen dos pasos de resolución bajo la aplicación de la regla general de resolución de (2.2), que se tomó sin cambios de la lógica proposicional.

Las circunstancias del siguiente ejemplo son algo más complejas. Suponemos que todos conocen a su propia madre y preguntamos si Henry conoce a alguien.

Con el símbolo de función “madre” y el predicado “sabe”, tenemos que derivar una contradicción de

$$(\text{conoce}(x, \text{madre}(x)))1 \quad (\neg \text{conoce}(\text{henry}, y))2.$$

Por el reemplazo x/henry, y/madre(henry) obtenemos el par de cláusulas contradictorias

$$(\text{sabe}(\text{enrique}, \text{madre}(\text{enrique})))1 \quad (\neg \text{sabe}(\text{enrique}, \text{madre}(\text{enrique})))2.$$

Este paso de reemplazo se llama unificación. Los dos literales son complementarios, lo que significa que son lo mismo aparte de sus signos. La cláusula vacía es ahora derivable con un paso de resolución, mediante el cual se ha demostrado que Henry no conocer a alguien (su madre). Definimos

Definición 3.7 Dos literales se llaman unificables si hay una sustitución σ para todas las variables lo que hace que los literales sean iguales. Tal σ se llama unificador. Un unificador se llama el unificador más general (MGU) si todos los demás unificadores pueden ser obtenido a partir de ella por sustitución de variables.

Ejemplo 3.6 Queremos unificar los literales $p(f(g(x)), y, z)$ y $p(u, u, f(u))$. Varios unificadores son

$\sigma_1 :$ $y/f(g(x)), z/f(f(g(x))), \sigma_2 : x/h(v), y/f(g(h(v))), z/f(f(g(h(v))))$, $u/f(g(x)), u/f(g(h(v)))$
 $\sigma_3 : x/h(h(v)), y/f(g(h(h(v))))$, $z/f(f(g(h(h(v))))$, $u/f(g(h(h(v))))$
 $\sigma_4 : x/h(a), y/f(g(h(a))), z/f(f(g(h(a))))$, $\sigma_5 : x/a, y/f(g(a)), z/f(f(g(a)))$, $u/f(g(h(a))), u/f(g(a))$

donde σ_1 es el unificador más general. Los otros unificadores resultan de σ_1 a través de la sustituciones $x/h(v)$, $x/h(h(v))$, $x/h(a)$, x/a .

Podemos ver en este ejemplo que durante la unificación de literales, los símbolos de predicado pueden tratarse como símbolos de función. Es decir, el literal se trata como un término. Las implementaciones de algoritmos de unificación procesan los argumentos de las funciones de forma secuencial. Los términos se unifican recursivamente sobre la estructura de términos. Los algoritmos de unificación más simples son muy rápidos en la mayoría de los casos. Sin embargo, en el peor de los casos, el tiempo de cálculo puede crecer exponencialmente con el tamaño de los términos. Porque para automatizado prueba que la abrumadora cantidad de intentos de unificación fracasan o son muy simples, en la mayoría de los casos el peor de los casos la complejidad no tiene un efecto dramático. La unificación más rápida los algoritmos tienen una complejidad casi lineal incluso en el peor de los casos [Bib82].

Ahora podemos dar la regla de resolución general para la lógica de predicados:

Definición 3.8 La regla de resolución para dos cláusulas en forma normal conjuntiva lee

$$\frac{(A_1 \quad \dots \quad A_m \quad B), (\neg B) \quad C_1 \quad \dots \quad C_n \quad \sigma(B) = \sigma(B')}{(\sigma(A_1) \quad \dots \quad \sigma(A_m) \quad \sigma(C_1) \quad \dots \quad \sigma(C_n))}, (3.6)$$

donde σ es la MGU de B y B' .

Teorema 3.5 La regla de resolución es correcta. Es decir, el resolutivo es una consecuencia semántica de las dos cláusulas principales.

Para Completitud, sin embargo, todavía necesitamos una pequeña adición, como se muestra en la siguiente ejemplo.

Ejemplo 3.7 La famosa paradoja de Russell dice : "Hay un barbero que afeita a todos los que no se afeitan a sí mismo". Esta declaración es contradictoria, lo que significa que es insatisfactoria. Deseamos mostrar esto con resolución. Formalizada en PL1, la paradoja dice

$$x \text{ afeita}(\text{barbero}, x) \quad \neg \text{afeita}(x, x)$$

y la transformación en forma de cláusula produce (vea el Ejercicio 3.6 en la página 55)

$$(\neg \text{afeitados}(\text{barbier}, x) \quad \neg \text{afeitados}(x, x))1 \quad (\text{afeitados}(\text{barbier}, x) \quad \text{afeitados}(x, x))2. \quad (3.7)$$

De estas dos cláusulas podemos derivar varias tautologías, pero ninguna contradicción. Por lo tanto, la resolución no es completa. Necesitamos aún una regla de inferencia adicional.

Definición 3.9 La factorización de una cláusula se logra mediante

$$\frac{(A1 \quad A2 \quad \dots \quad An) \sigma \quad (A1) = \sigma \quad (A2) \quad (\sigma \quad (A2))}{\dots \quad \sigma \quad (An)},$$

donde σ es la MGU de $A1$ y $A2$.

Ahora se puede derivar una contradicción de (3.7)

$$\begin{aligned} \text{Fak}(1, \sigma : x/\text{barbero}) : (\neg \text{afeitado}(\text{barbero}, \text{barbero}))3 \\ \text{Fak}(2, \sigma : x/\text{barbero}) : (\text{afeitado}(\text{barbero}, \text{barbero}))4 \\ \text{Res}(3, 4) : ()5 \end{aligned}$$

y afirmamos:

Teorema 3.6 La regla de resolución (3.6) junto con la regla de factorización (3.9) es una refutación completa. Es decir, mediante la aplicación de pasos de factorización y resolución, la cláusula vacía puede derivarse de cualquier fórmula insatisfactoria en forma normal conjuntiva.

3.5.1 Estrategias de resolución

Si bien la resolución completa es importante para el usuario, la búsqueda de una prueba puede ser muy frustrante en la práctica. La razón de esto es el inmenso espacio de búsqueda combinatoria. Incluso si hay muy pocos pares de cláusulas en KB $\neg Q$ al principio, el probador genera una nueva cláusula con cada paso de resolución, lo que aumenta el número de posibles pasos de resolución en la siguiente iteración. Por lo tanto, durante mucho tiempo se ha intentado reducir el espacio de búsqueda utilizando estrategias especiales, preferiblemente sin perder la integridad. Las estrategias más importantes son las siguientes.

La resolución unitaria prioriza los pasos de resolución en los que una de las dos cláusulas consta de un solo literal, denominado cláusula unitaria. Esta estrategia preserva la integridad y conduce en muchos casos, pero no siempre, a una reducción del espacio de búsqueda. Por lo tanto, es un proceso heurístico (ver Secc. 6.3).

Se obtiene una reducción garantizada del espacio de búsqueda por aplicación del conjunto de estrategia de soporte. Aquí un subconjunto de KB $\neg Q$ se define como el conjunto de apoyo (SOS). Cada paso de resolución debe incluir una cláusula del SOS, y el resolutivo se agrega al SOS. Esta estrategia es incompleta. Se completa cuando se asegura que el conjunto de cláusulas es satisficible sin el SOS (ver Ejercicio 3.7 en la página 55). La consulta negada $\neg Q$ se usa a menudo como SOS inicial.

En la resolución de entrada, una cláusula del conjunto de entrada KB $\neg Q$ debe estar involucrada en cada paso de resolución. Esta estrategia también reduce el espacio de búsqueda, pero a costa de la exhaustividad.

Con la regla de los literales puros se pueden eliminar todas las cláusulas que contienen literales para los cuales no hay literales complementarios en otras cláusulas. Esta regla reduce el espacio de búsqueda y es completa, por lo que la utilizan prácticamente todos los probadores de resolución.

Si los literales de una cláusula K1 representan un subconjunto de los literales de la cláusula K2, entonces se puede eliminar K2. Por ejemplo, la cláusula

(lloviendo (hoy) $\rightarrow$ street_wet (hoy))

es redundante si street_wet(today) ya es válido. Este importante paso de reducción se llama subsunción. La subsunción también es completa.

3.5.2 Igualdad

La igualdad es una causa especialmente inconveniente del crecimiento explosivo del espacio de búsqueda. Si agregamos (3.1) en la página 36 y los axiomas de igualdad formulados en (3.2) en la página 36 a la base de conocimientos, entonces la cláusula de simetría $\neg x = y \rightarrow y = x$ se puede unificar con cada ecuación positiva o negativa, por ejemplo. Esto conduce a la derivación de nuevas cláusulas y ecuaciones sobre las que se pueden aplicar de nuevo los axiomas de igualdad, y así sucesivamente. Los axiomas de transitividad y sustitución tienen consecuencias similares. Debido a esto, se han desarrollado reglas de inferencia especiales para la igualdad que funcionan sin axiomas de igualdad explícitos y, en particular, reducen el espacio de búsqueda. demodulación,

por ejemplo, permite la sustitución de un término t_2 por t_1 , si existe la ecuación $t_1 = t_2$. Un la ecuación $t_1 = t_2$ se aplica mediante unificación a un término t de la siguiente manera:

$$\frac{t_1 = t_2, \quad (\dots t \dots), \sigma(t_1) = \sigma(t)}{(\dots \sigma(t_2) \dots)}.$$

Algo más general es la paramodulación, que trabaja con ecuaciones condicionales [Bib82, Lov78].

La ecuación $t_1 = t_2$ permite la sustitución del término t_1 por t_2 así como el sustitución de t_2 por t_1 . Por lo general, no tiene sentido revertir una sustitución que ya ha sido llevado a cabo. Por el contrario, las ecuaciones se utilizan con frecuencia para simplificar términos. Por lo tanto, a menudo se usan en una sola dirección. Ecuaciones que solo se utilizan en una dirección se llaman ecuaciones dirigidas. El procesamiento eficiente de ecuaciones dirigidas se logra mediante los llamados sistemas de reescritura de términos. Para fórmulas con muchos ecuaciones existen probadores especiales de igualdad.

3.6 Demostradores automáticos de teoremas

Las implementaciones de cálculos de prueba en computadoras se denominan probadores de teoremas. A lo largo de con probadores especializados para subconjuntos de PL1 o aplicaciones especiales, existen hoy toda una línea de probadores automatizados para la lógica de predicados completa y la lógica de orden superior, de los cuales solo algunos se discutirán aquí. Una visión general de los más importantes Los sistemas se pueden encontrar en [McC].

Uno de los probadores de resolución más antiguos se desarrolló en el oratorio Argonne National Lab en Chicago. Basado en los primeros desarrollos que comenzaron en 1963, Otter [Kal01], fue creada en 1984. Sobre todo, Otter se aplicó con éxito en áreas especializadas de las matemáticas, como se puede aprender en su página de inicio:

"Actualmente, la principal aplicación de Otter es la investigación en álgebra abstracta y lógica formal. Otter y sus predecesores se han utilizado para responder muchas preguntas abiertas en las áreas de semigrupos finitos, álgebra booleana ternaria, cálculo lógico, lógica combinatoria, teoría de grupos, teoría de la red y geometría algebraica".

Varios años más tarde, la Universidad de Tecnología de Múnich creó el probador de alto rendimiento SETHEO [LSBB92] basado en la rápida tecnología PROLOG. Con el objetivo de alcanzar un rendimiento aún mayor, se desarrolló una implementación para computadores paralelos bajo el nombre de PARTHEO. Resultó que no valía la pena usar hardware especial en probadores de teoremas, como también es el caso en otras áreas. de IA, porque estas computadoras son superadas muy rápidamente por procesadores más rápidos y algoritmos más inteligentes. Múnich es también el lugar de nacimiento de E [Sch02], un probador de ecuaciones moderno galardonado, con el que nos familiarizaremos en el próximo ejemplo. En la página de inicio de E se pueden leer las siguientes características compactas e irónicas.

zation, cuya segunda parte, por cierto, se aplica a todos los probadores automáticos existentes hoy.

"E es un probador de teoremas puramente ecuacional para la lógica de cláusulas. Eso significa que es un programa que puede incluir una especificación matemática (en lógica de cláusulas con igualdad) y una hipótesis, y que luego se ejecutará para siempre, consumiendo todos los recursos de su máquina. Muy ocasionalmente encontrará una prueba para la hipótesis y te lo dirá :-).".

Aparentemente, encontrar pruebas para proposiciones verdaderas es tan difícil que la búsqueda tiene éxito muy rara vez, o solo después de mucho tiempo, si es que lo logra. vamos a entrar esto con más detalle en el Cap. 4. Aquí debe mencionarse, sin embargo, que no sólo computadoras, pero también la mayoría de las personas tienen problemas para encontrar pruebas formales estrictas.

Aunque evidentemente las computadoras por sí solas son en muchos casos incapaces de encontrar una prueba, lo siguiente mejor es construir sistemas que funcionen de forma semiautomática. y permitir una estrecha cooperación con el usuario. De este modo, el ser humano puede aplicar mejor su conocimiento de dominios de aplicación especiales y tal vez limitar la búsqueda de la prueba. Uno de los probadores interactivos más exitosos para la lógica de predicados de orden superior es Isabelle [NPW02], un producto común de la Universidad de Cambridge y la Universidad de Tecnología, Múnich.

Cualquiera que busque un probador de alto rendimiento debe consultar los resultados actuales de la CASC (CADE ATP System Competition) [SS06].¹ Aquí encontramos que el ganador de 2001 a 2006 en las categorías PL1 y cláusula forma normal fue El probador Vampire de Manchester, que funciona con una variante de resolución y un especial acercamiento a la igualdad. El sistema Waldmeister del Instituto Max Planck en Saar brücken ha sido líder durante años en la demostración de la igualdad.

Las muchas posiciones principales de los sistemas alemanes en CASC muestran que la investigación alemana grupos en el área de demostración automática de teoremas están jugando un papel de liderazgo, hoy así como en el pasado.

3.7 Ejemplos matemáticos

Ahora deseamos demostrar la aplicación de un probador automatizado con el probador E [Sch02] mencionado anteriormente. E es un probador de igualdad especializado que reduce en gran medida el espacio de búsqueda a través de un tratamiento optimizado de la igualdad.

Queremos demostrar que los elementos neutrales a la izquierda y a la derecha en un semigrupo son iguales. Primero formalizamos la reclamación paso a paso.

¹CADE es la "Conferencia sobre Deducción Automatizada" [CAD] anual y ATP significa "Probador de Teorema Automatizado".

Definición 3.10 Una estructura $(M, \cdot)$ que consta de un conjunto M con una operación interna biplaza " $\cdot$ " se denomina semigrupo si la ley de asociatividad

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

sostiene. Un elemento $e \in M$ se llama neutral a la izquierda (neutral a la derecha) si $xe = x$ ($xx = x$).

Queda por demostrar que

Teorema 3.7 Si un semigrupo tiene un elemento neutral a la izquierda el y un elemento neutral a la derecha er , entonces $el = er$.

Primero demostramos el teorema semi-formalmente por razonamiento matemático intuitivo. Claramente se cumple para todo $x \in M$ que

$$el \cdot x = x \tag{3.8}$$

y

$$x \cdot er = x. \tag{3.9}$$

Si establecemos $x = er$ en (3.8) y $x = el$ en (3.9), obtenemos las dos ecuaciones $el \cdot er = er$ y $el \cdot er = el$. Uniendo estas dos ecuaciones se obtiene

$$el = el \cdot er = er,$$

que queremos probar. En el último paso, dicho sea de paso, usamos el hecho de que la igualdad es simétrica y transitiva.

Antes de aplicar el probador automatizado, realizamos la prueba de resolución manualmente. Primero formalizamos la consulta negada y la base de conocimiento KB, que consta de los axiomas como cláusulas en forma normal conjuntiva:

$(\neg el = er)$ 1	consulta negada
$(m(m(x, y), z) = m(x, m(y, z)))$ 2	
$(m(el, x) = x)$ 3	
$(m(x, er) = x)$ 4	

Axiomas de	
igualdad:	
$(x = x)$ 5	(reflexividad)
$(\neg x = y \rightarrow y = x)$ 6	(simetría)
$(\neg x = y \rightarrow \neg y = z \rightarrow x = z)$ 7	(transitividad)
$(\neg x = y \rightarrow m(x, z) = m(y, z))$ 8	sustitución en m
$(\neg y = z \rightarrow m(x, y) = m(x, z))$ 9	sustitución en m ,

donde la multiplicación está representada por el símbolo de función de dos lugares m. Los axiomas de igualdad se formularon de manera análoga a (3.1) en la página 36 y (3.2) en la página 36. Una prueba de resolución simple tiene la forma

$$\begin{aligned} &\text{Res}(3, 6, x6/m(el, x3), y6/x3): \text{Res}(7, (x = m(el, x))10 \\ &10, x7/x10, y7/m(el, x10)): (\neg m(el, x) = z \quad x = z)11 \\ &\text{Res}(4, 11, x4/el, x11/er, z11/el): (er = el)12 \\ &\text{Res}(1, 12,): (). \end{aligned}$$

Aquí, por ejemplo, Res(3, 6, x6/m(el, x3), y6/x3) significa que en la resolución de la cláusula 3 con la cláusula 6, la variable x de la cláusula 6 se reemplaza por m(el, x3) con variable x de la cláusula 3. Análogamente, y de la cláusula 6 se reemplaza por x de cláusula 3.

Ahora queremos aplicar el probador E al problema. Las cláusulas se transforman en el lenguaje de forma normal de la cláusula LOP a través del mapeo

$$(\neg A1 \quad \dots \quad \neg Am \quad B1 \quad \dots \quad Bn) \rightarrow B1; \dots; Bn < \neg A1, \dots, Am.$$

La sintaxis de LOP representa una extensión de la sintaxis de PROLOG (ver Secc. 5) para Cláusulas no Horn. Así obtenemos como archivo de entrada para E

```
<- el = er.                                     % consulta
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X        % asociatividad de m
m(X,er) = X                                     % elemento neutral a la izquierda de m
                                                % elemento neutral a la derecha de m
```

donde la igualdad está modelada por el símbolo de predicado gl. Llamar al probador entrega

```
unixprompt> prueba electrónica halbgr1a.lop
# Estado del problema determinado, construyendo objeto de prueba
# Comienza la evidencia del estado del problema
0 : [--igual(el, er)] : inicial
1 : [++igual(m(el,X1), X1)] : inicial
2: [++igual(m(X1,er), X1)] : inicial
3 : [++igual(el, er)] : pm(2,1)
4 : [--igual(el, el)] : rw(0,3)
5 : [] : cn(4)
6 : [] : 5 : {prueba}
# Termina la evidencia del estado del problema
```

Los literales positivos se identifican con ++ y los negativos con --. En las líneas 0 a 4, marcado con inicial, las cláusulas de los datos de entrada se enumeran de nuevo. pm(a,b) representa un paso de resolución entre la cláusula ay la cláusula b. Vemos que la prueba

encontrado por E es muy similar a la prueba creada manualmente. Porque nosotros explícitamente modelar la igualdad por el predicado `gl`, las fuerzas particulares de E no vienen en juego. Ahora omitimos los axiomas de igualdad y obtenemos

```
<- el = er.                                     % Consulta
m(m(X,Y),Z) = m(X,m(Y,Z)) . m(el,X) = X        % Asociación
m(X,er) = X                                     % enlaces neutros El. v m
                                                % neutrales neutrales El. v m
```

como entrada para el probador.

La prueba también se vuelve más compacta. Vemos en la siguiente salida del probar que la prueba consiste esencialmente en un solo paso de inferencia sobre las dos cláusulas 1 y 2 relevantes.

```
unixprompt> prueba electrónica halbgr1a.lop
# Estado del problema determinado, construyendo objeto de prueba
# Comienza la evidencia del estado del problema
0 : [--igual(el, er)] : inicial
1 : [++igual(m(el,X1), X1)] : inicial
2: [++igual(m(X1,er), X1)] : inicial
3 : [++igual(el, er)] : pm(2,1)
4 : [--igual(el, el)] : rw(0,3)
5 : : cn(1)
6 : [] : 5 : {prueba}
# Termina la evidencia del estado del problema
```

El lector ahora podría echar un vistazo más de cerca a las capacidades de E (Ejercicio 3.9 sobre página 55).

3.8 Aplicaciones

En matemáticas, los probadores automáticos de teoremas se utilizan para ciertas tareas especializadas. Por ejemplo, el importante teorema de los cuatro colores de la teoría de grafos se demostró por primera vez en 1976 con la ayuda de un experimentador especial. Sin embargo, los probadores automáticos todavía juegan un papel menor. papel en las matemáticas.

Por otro lado, en los inicios de la IA, la lógica de predicados era de gran importancia. para el desarrollo de sistemas expertos en aplicaciones prácticas. Debido a sus problemas modelando la incertidumbre (ver Secc. 4.4), los sistemas expertos en la actualidad se desarrollan con mayor frecuencia utilizando otros formalismos.

Hoy la lógica juega un papel cada vez más importante en las tareas de verificación. Automático La verificación de programas es actualmente un área de investigación importante entre la IA y la ingeniería de software. Los sistemas de software cada vez más complejos ahora están asumiendo tareas

de cada vez más responsabilidad y relevancia en materia de seguridad. Aquí es deseable una prueba de ciertas características de seguridad de un programa. Tal prueba no puede lograrse mediante la prueba de un programa terminado, ya que, en general, es imposible aplicar un programa a todas las entradas posibles. Por lo tanto, este es un dominio ideal para sistemas de inferencia generales o incluso especializados. Entre otras cosas, hoy en día se utilizan protocolos criptográficos cuyas características de seguridad se han verificado automáticamente [FS97, Sch01]. Otro desafío para el uso de probadores automatizados es la síntesis de software y hardware. Para ello, por ejemplo, los probadores deberían apoyar al ingeniero de software en la generación de programas a partir de especificaciones.

La reutilización de software también es de gran importancia para muchos programadores en la actualidad. El programador busca un programa que tome datos de entrada con ciertas propiedades y calcule un resultado con las propiedades deseadas. Un algoritmo de clasificación acepta datos de entrada con entradas de un determinado tipo de datos y, a partir de estos, crea una permutación de estas entradas con la propiedad de que cada elemento es menor o igual que el elemento siguiente. El programador primero formula una especificación de la consulta en PL1 que consta de dos partes. La primera parte PREQ comprende las condiciones previas, que deben cumplirse antes de que se aplique el programa deseado. La segunda parte POSTQ contiene las condiciones posteriores, que deben cumplirse después de aplicar el programa deseado.

En el siguiente paso, se debe buscar en una base de datos de software módulos que cumplan con estos requisitos. Para verificar esto formalmente, la base de datos debe almacenar una descripción formal de las condiciones previas PREM y postcondiciones POSTM para cada módulo M. Una suposición sobre las capacidades de los módulos es que las condiciones previas del módulo se derivan de las condiciones previas de la consulta. debe sostener que

$$\text{PREQ} \quad \text{PREM}.$$

Todas las condiciones que se requieren como requisito previo para la aplicación del módulo M deben aparecer como condiciones previas en la consulta. Si, por ejemplo, un módulo en la base de datos solo acepta listas de números enteros, entonces las listas de números enteros como entrada también deben aparecer como condiciones previas en la consulta. Un requisito adicional en la consulta de que, por ejemplo, solo aparezcan números pares, no genera problema.

Además, debe cumplirse para las poscondiciones que

$$\text{POSTM} \quad \text{POSTQ}.$$

Es decir, luego de la aplicación del módulo, se deben cumplir todos los atributos que requiere la consulta. Ahora mostramos la aplicación de un probador de teoremas a esta tarea en un ejemplo de [Sch01].

Ejemplo 3.8 VDM-SL, el lenguaje de especificación del método de desarrollo de Viena, se usa a menudo como lenguaje para la especificación de condiciones previas y posteriores. Suponga que en la base de datos del software está disponible la descripción de un módulo ROTATE, que mueve el primer elemento de la lista al final de la lista. Estamos buscando un módulo SHUFFLE, que crea una permutación arbitraria de la lista. Las dos especificaciones dicen

GIRAR($l : \text{Lista}$) $l' : \text{Lista}$ anterior a la verdad correo $(l = [] \mid l' = [])$ $(l = [] \mid l' = (\text{cola } l)[\text{cabeza } l])$	SHUFFLE($x : \text{Lista}$) $x' : \text{Lista}$ anterior a la verdad puesto $i : \text{Artículo}$ $(x1, x2 : \text{Lista} \cdot x = x1[i] \cdot x2$ $y1, y2 : \text{Lista} \cdot x' = y1[i] \cdot y2)$

Aquí representa la concatenación de listas, y “.” separa los cuantificadores con sus variables del resto de la fórmula. Las funciones “cabeza l” y “cola l” elija el primer elemento y el resto de la lista, respectivamente. La especificación de SHUFFLE indica que cada elemento de la lista i que estaba en la lista (x) antes de la aplicación de SHUFFLE debe estar en el resultado (x') después de la aplicación, y viceversa. Ahora debe demostrarse que la fórmula (PREQ PREM) (POSTM POSTQ) es una consecuencia de la base de conocimiento que contiene una descripción del Lista de tipos de datos. Las dos especificaciones VDM-SL producen la tarea de prueba

$$\begin{aligned}
 & l, l' : \text{Lista} \cdot (l = x \mid l' = x) \wedge (w = w') \\
 & (l = x \mid l' = x) \wedge ((l = [] \mid l' = []) \vee (l = [] \mid l' = (\text{tl } l)[\text{hd } l])) \\
 & i : \text{Artículo} \cdot (x1, x2 : \text{Lista} \cdot x = x1[i] \cdot x2 \wedge y1, y2 : \text{Lista} \cdot x' = y1[i] \cdot y2)),
 \end{aligned}$$

que luego se puede probar con el probador SETHEO.

En los próximos años, la web semántica probablemente representará una aplicación importante de PL1. Se supone que el contenido de la World Wide Web se vuelve interpretable no solo para las personas, sino también para las máquinas. Con este fin, los sitios web están siendo equipados con una descripción de su semántica en un lenguaje de descripción formal. La búsqueda de la información en la web será significativamente más efectiva que hoy, donde esencialmente solo se pueden buscar bloques de construcción de texto.

Los subconjuntos decidibles de la lógica de predicados se utilizan como lenguajes de descripción. El desarrollo de cálculos eficientes para el razonamiento es muy importante y está estrechamente relacionado a los lenguajes de descripción. Una consulta para un futuro motor de búsqueda que opere semánticamente podría leer (informalmente): ¿Dónde en Suiza el próximo domingo en elevaciones bajo 2000 metros habrá buen tiempo y pistas de esquí óptimamente preparadas? Para responder a tal pregunta, se requiere un cálculo que sea capaz de trabajar muy rápidamente.

en grandes conjuntos de hechos y reglas. Aquí, los términos de funciones anidadas complejas son menos importantes.

Como marco descriptivo básico, el World Wide Web Consortium desarrolló el lenguaje RDF (Resource Description Framework). Basado en RDF, el lenguaje significativamente más poderoso OWL (Web Ontology Language) permite la descripción de relaciones entre objetos y clases de objetos, de manera similar a PL1 [SET09].

Las ontologías son descripciones de relaciones entre posibles objetos.

Una dificultad al construir una descripción de los innumerables sitios web es el gasto de trabajo y también la verificación de la corrección de las descripciones semánticas. Aquí los sistemas de aprendizaje automático para la generación automática de descripciones pueden ser de gran ayuda. Luis von Ahn de la Universidad Carnegie Mellon [vA06] introdujo un uso interesante de la generación “automática” de semántica en la web. Desarrolló juegos de computadora en los que se supone que los jugadores, distribuidos por la red, deben describir en colaboración imágenes con palabras clave. Así se asigna semánticamente a las imágenes de forma divertida y sin coste alguno. El lector puede probar los juegos y escuchar un discurso del inventor en el Ejercicio 3.10 de la página 55.

3.9 Resumen

Hemos proporcionado los fundamentos, términos y procedimientos más importantes de la lógica de predicados y hemos demostrado que incluso una de las tareas intelectuales más difíciles, a saber, la demostración de teoremas matemáticos, puede automatizarse. Los probadores automatizados pueden emplearse no solo en matemáticas, sino también, en particular, en tareas de verificación en informática. Sin embargo, para el razonamiento cotidiano, la lógica de predicados en la mayoría de los casos no es adecuada. En los capítulos siguientes y siguientes mostramos sus puntos débiles y algunas alternativas modernas interesantes. Además, mostraremos en el Cap. 5 que uno puede programar elegantemente con lógica y sus extensiones procedimentales.

Cualquier persona interesada en la lógica de primer orden, la resolución y otros cálculos para probadores automatizados encontrará buenas instrucciones avanzadas en [New00, Fit96, Bib82, Lov78, CL73]. Las referencias a los recursos de Internet se pueden encontrar en el sitio web de este libro.

3.10 Ejercicios

Ejercicio 3.1 Sean dados el predicado de tres lugares “niño” y el predicado de un lugar “femenino” del Ejemplo 3.2 en la página 35. Defina: (a) Un predicado de un solo

lugar “masculino”. (b) Un predicado de dos

lugares “padre” y “madre”. (c) Un predicado de dos lugares

“hermanos”. (d) Un predicado “padres(x, y, z)”,

que es verdadero si y solo si x es el padre y y es la madre de z .

(e) Un predicado “tío(x, y)”, que es verdadero si y sólo si x es el tío de y (use los predicados que ya han sido definidos).

(f) Un predicado de dos lugares “antepasado” con el significado: los antepasados son padres, abuelos, etc. de muchas generaciones arbitrarias.

Ejercicio 3.2 Formalice los siguientes enunciados en lógica de predicados: (a) Toda persona tiene un padre y una madre. (b) Algunas personas tienen hijos. (c) Todas las aves vuelan.

- (d) Hay un animal que come (algunos) animales que comen granos.
 (e) Todo animal come plantas o animales herbívoros que son mucho más pequeños que sí mismo.

Ejercicio 3.3 Adapte el Ejercicio 3.1 de la página 54 usando símbolos de función de un lugar e igualdad en lugar de "padre" y "madre".

Ejercicio 3.4 Dar axiomas de lógica de predicados para la relación de dos lugares "<" como un orden total. Para un pedido total debemos tener (1) Dos elementos cualesquiera son comparables. (2) Es simétrico. (3) Es transitivo.

Ejercicio 3.5 Unifique (si es posible) los siguientes términos y dé la MGU y los términos resultantes.

- (a) $p(x, f(y))$, $p(f(z), u)$ (b) $p(x, f(x))$, $p(y, y)$ (c)
 $x = 4 - 7 \cdot x$, $\cos y = z$ (d) $x < 2 \cdot x$, $3 < 6$ (e) $q(f(x, y, z), f(g(w, w), g(x, x), g(y, y)))$, $q(u, u)$

Ejercicio 3.6

- (a) Transforme la paradoja de Russell del Ejemplo 3.7 en la página 45 en CNF. (b) Demuestre que la cláusula vacía no se puede derivar usando la resolución sin factorización de (3.7) en la página 45. Trate de entender esto intuitivamente.

Ejercicio 3.7

- (a) ¿Por qué la resolución con el conjunto de estrategia de apoyo es incompleta? (b) Justifique (sin probar) por qué el conjunto de estrategias de apoyo se vuelve completo si $(KB \neg Q) \setminus \text{SOS}$ es satisfactorio.
 (c) ¿Por qué es completa la resolución con la regla literal pura?

Ejercicio 3.8 Formalizar y demostrar con resolución que en un semigrupo con al menos dos elementos diferentes a, b , un elemento e neutro por la izquierda y un elemento n nulo por la izquierda, estos dos elementos tienen que ser diferentes, es decir, que $n \neq e$. Utilice la demodulación, que permite el reemplazo de "similar con similar".

Ejercicio 3.9 Obtenga el probador del teorema E [Sch02] u otro probador y pruebe las siguientes afirmaciones. Compara estas demostraciones con las del texto. (a) La afirmación del Ejemplo 2.3 en la página 24. (b) La paradoja de Russell del Ejemplo 3.7 en la página 45. (c) La afirmación del Ejercicio 3.8.

Ejercicio 3.10 Prueba los juegos www.espgame.org y www.peekaboomb.org, que ayudan a asignar semántica a las imágenes en la web. Escuche la conferencia sobre computación humana de Luis von Ahn en: <http://video.google.de/videoplay?docid=-8246463980976635143>.

4.1 El problema del espacio de búsqueda

Como ya se mencionó en varios lugares, en la búsqueda de una prueba casi siempre hay muchas (dependiendo del cálculo, potencialmente infinitas) posibilidades para la aplicación de reglas de inferencia en cada paso. El resultado es el crecimiento explosivo antes mencionado del espacio de búsqueda (Fig. 4.1 en la página 58). En el peor de los casos, se deben probar todas estas posibilidades para encontrar la prueba, lo que generalmente no es posible en un tiempo razonable.

Si comparamos probadores automáticos o sistemas de inferencia con matemáticos o expertos humanos que tienen experiencia en dominios especiales, hacemos observaciones interesantes. Por un lado, los matemáticos experimentados pueden demostrar teoremas que están fuera del alcance de los probadores automáticos. Por otro lado, los probadores automatizados realizan decenas de miles de inferencias por segundo. Un ser humano, por el contrario, realiza tal vez una inferencia por segundo. Aunque los expertos humanos son mucho más lentos a nivel de objeto (es decir, en la realización de inferencias), aparentemente resuelven problemas difíciles mucho más rápido.

Hay varias razones para esto. Los humanos usamos cálculos intuitivos que funcionan en un nivel superior y, a menudo, llevamos a cabo muchas de las inferencias simples de un probador automatizado en un solo paso. Además, usamos lemas, es decir, fórmulas verdaderas derivadas que ya conocemos y, por lo tanto, no necesitamos volver a probarlas cada vez. Mientras tanto, también hay probadores de máquinas que funcionan con tales métodos. Pero ni siquiera ellos pueden competir todavía con los expertos humanos.

Otra ventaja mucho más importante de nosotros los humanos es la intuición, sin la cual no podríamos resolver ningún problema difícil [PS09]. El intento de formalizar la intuición causa problemas. La experiencia en proyectos de IA aplicada muestra que en dominios complejos como la medicina (consulte la Sección 7.3) o las matemáticas, la mayoría de los expertos no pueden formular verbalmente este metaconocimiento intuitivo, y mucho menos formalizarlo. Por lo tanto, no podemos programar este conocimiento o integrarlo en cálculos en forma de heurística. Las heurísticas son métodos que en muchos casos pueden simplificar o acortar en gran medida el camino hacia la meta, pero en algunos casos (normalmente en raras ocasiones) pueden alargar en gran medida el



Fig. 4.1 Posibles consecuencias de la explosión de un espacio de búsqueda

a la meta La búsqueda heurística es importante no solo para la lógica, sino en general para la resolución de problemas en IA y, por lo tanto, se tratará a fondo en el capítulo 1. 6.

Un enfoque interesante, que se ha seguido desde alrededor de 1990, es la aplicación de técnicas de aprendizaje automático al aprendizaje de heurísticas para dirigir la búsqueda de sistemas de inferencia, que describiremos brevemente a continuación. Un probador de resolución tiene, durante la búsqueda de una prueba, cientos o más posibilidades de pasos de resolución en cada paso, pero solo unos pocos conducen a la meta. Sería ideal si el probador pudiera preguntarle a un oráculo qué dos cláusulas debería usar en el siguiente paso para encontrar rápidamente la prueba. Hay intentos de construir tales módulos de dirección de prueba, que evalúan las diversas alternativas para el siguiente paso y luego eligen la alternativa con la mejor calificación. En el caso de resolución, la calificación de las cláusulas disponibles podría ser computada por una función que calcula un valor basado en el número de literales positivos, la complejidad de los términos, etc., para cada par de cláusulas resolubles.

¿Cómo se puede implementar esta función? Debido a que este conocimiento es "intuitivo", el programador no está familiarizado con él. En cambio, uno intenta copiar la naturaleza y usa algoritmos de aprendizaje automático para aprender de pruebas exitosas [ESS89, SE90]. Los atributos de todos los pares de cláusulas que participan en los pasos de resolución exitosos se almacenan como positivos y los atributos de todas las resoluciones fallidas se almacenan como negativos.

Luego, utilizando estos datos de entrenamiento y un sistema de aprendizaje automático, se genera un programa que puede calificar pares de cláusulas de manera heurística (consulte la Sección 9.5).

Un enfoque diferente y más exitoso para mejorar el razonamiento matemático es seguido de sistemas interactivos que operan bajo el control del usuario. aquí uno podría nombrar programas de álgebra computacional como Mathematica, Maple o Maxima, que puede llevar a cabo automáticamente difíciles manipulaciones matemáticas simbólicas. La búsqueda de la prueba, sin embargo, se deja totalmente al ser humano. Isabelle [NPW02], el probador interactivo antes mencionado, brinda claramente más apoyo durante la demostración. buscar. Actualmente existen varios proyectos, como Omega [SB04] y MKM,¹ para el desarrollo de sistemas para apoyar a los matemáticos durante las demostraciones.

En resumen, se puede decir que, debido al problema del espacio de búsqueda, automatizado los probadores de hoy solo pueden probar teoremas relativamente simples en dominios especiales con pocos axiomas.

4.2 Decidibilidad e incompletitud

La lógica de predicados de primer orden proporciona una poderosa herramienta para la representación del conocimiento y el razonamiento. Sabemos que existen cálculos correctos y completos y probadores de teoremas. Cuando se prueba un teorema, es decir, un enunciado verdadero, tal probador es muy útil porque, debido a la completitud, uno sabe después de un tiempo finito que la declaración realmente es verdad ¿Qué pasa si la declaración no es verdadera? El teorema de completitud (Teorema 3.3 en la página 41) no responde a esta pregunta.² Específicamente, no hay proceso que puede probar o refutar cualquier fórmula de PL1 en tiempo finito, pues sostiene que

Teorema 4.1 El conjunto de fórmulas válidas en la lógica de predicados de primer orden es semidecidible.

Este teorema implica que existen programas (probadores de teoremas) que, dada una fórmula verdadera (válida) como entrada, determine su verdad en un tiempo finito. Si la fórmula no es válido, sin embargo, puede suceder que el probador nunca se detenga. (El lector puede lidiar con esta pregunta en el Ejercicio 4.1 de la página 65.) La lógica proposicional es decidible porque el método de la tabla de verdad proporciona todos los modelos de una fórmula en un tiempo finito. Evidentemente, la lógica de predicados con cuantificadores y símbolos de funciones anidadas es un lenguaje algo demasiado poderoso para ser decidible.

Por otro lado, la lógica de predicados no es lo suficientemente poderosa para muchos propósitos. A menudo se desea hacer enunciados sobre conjuntos de predicados o funciones. Este no funciona en PL1 porque solo conoce cuantificadores para variables, pero no para predicados o funciones.

Kurt Gödel mostró, poco después de su teorema de completitud para PL1, que la completitud se pierde si extendemos PL1 aunque sea mínimamente para construir una lógica de orden superior. Una lógica de primer orden solo puede cuantificar sobre variables. Una lógica de segundo orden también puede

¹www.mathweb.org/mathweb/demo.html.

²Solo este caso es especialmente importante en la práctica, porque si ya sé que una declaración es verdadera, Ya no necesito un probador.

cuantificar sobre fórmulas de primer orden, y una lógica de tercer orden puede cuantificar sobre fórmulas de segundo orden. Incluso agregando solo el axioma de inducción para los números naturales, la lógica queda incompleta. La afirmación "Si un predicado $p(n)$ se cumple para n , entonces $p(n + 1)$ también se cumple", o

$$p(n) \rightarrow p(n + 1)$$

es una proposición de segundo orden porque cuantifica sobre un predicado. Gödel demostró el siguiente teorema:

Teorema 4.2 (Teorema de incompletitud de Gödel) Todo sistema de axiomas para los números naturales con suma y multiplicación (aritmética) es incompleto. Es decir, hay enunciados verdaderos en aritmética que no son demostrables.

La prueba de Gödel funciona con lo que se llama Gödelización, en la que cada fórmula aritmética se codifica como un número. Obtiene un número de Gödel único. Gödelización ahora se usa para formular la proposición

$$F = \text{"No soy demostrable"}$$

en el lenguaje de la aritmética. Esta fórmula es verdadera por la siguiente razón. Suponga que F es falsa. Entonces podemos probar F y por lo tanto mostrar que F no es demostrable. Esto es una contradicción. Por lo tanto, F es verdadera y, por lo tanto, no demostrable.

El trasfondo más profundo de este teorema es que las teorías matemáticas (sistemas de axiomas) y, más generalmente, los lenguajes se vuelven incompletos si el lenguaje se vuelve demasiado poderoso. Un ejemplo similar es la teoría de conjuntos. Este lenguaje es tan poderoso que uno puede formular paradojas con él. Son enunciados que se contradicen a sí mismos, como el enunciado que ya conocemos del Ejemplo 3.7 de la página 45 sobre los barberos que afeitan a los que no se afeitan a sí mismos (ver Ejercicio 4.2 de la página 65).³ El dilema consiste en que con los lenguajes que son lo suficientemente potentes como para describir matemáticas y aplicaciones interesantes, estamos pasando de contrabando contradicciones e incompletudes por la puerta de atrás. Sin embargo, esto no significa que las lógicas de orden superior sean totalmente inadecuadas para los métodos formales. Ciertamente, existen sistemas formales, así como probadores para lógicas de orden superior.

4.3 El pingüino volador

Con un ejemplo sencillo demostraremos un problema fundamental de lógica y posibles enfoques de solución. Dadas las declaraciones

³En [Wie] se pueden encontrar muchas más paradojas lógicas.



Fig. 4.2 El pingüino volador Piolín

1. Piolín es un pingüino
2. Los pingüinos son pájaros
3. Los pájaros pueden volar

Formalizado en PL1, la base de conocimientos KB da como resultado:

```
pingüino (tweety)
pingüino (x)   pájaro (x) pájaro
(x)   mosca (x)
```

De ahí (por ejemplo con resolución) se puede derivar $\text{fly}(\text{tweety})$ (Fig. 4.2).⁴ Evidentemente, la formalización de los atributos de vuelo de los pingüinos es insuficiente. Probamos el enunciado adicional Los pingüinos no pueden volar, es decir

```
pingüino(x)   ¬mosca(x)
```

A partir de ahí se puede derivar $\neg \text{fly}(\text{tweety})$. Pero $\text{fly}(\text{tweety})$ sigue siendo cierto. Por lo tanto, la base de conocimientos es inconsistente. Aquí notamos una característica importante de la lógica, a saber, la monotonía. Aunque declaramos explícitamente que los pingüinos no pueden volar, todavía se puede derivar el sitio opuesto.

Definición 4.1 Una lógica se llama *monotónica* si, para una base de conocimiento arbitraria KB y una fórmula arbitraria ϕ , el conjunto de fórmulas derivables de KB es un subconjunto de las fórmulas derivables de KB $\cup \phi$.

⁴La ejecución formal de esta prueba simple y la siguiente puede dejarse al lector (Ejercicio 4.3 de la página 65).

Si se amplía un conjunto de fórmulas, entonces, después de la ampliación, todos los enunciados derivables anteriormente aún pueden probarse, y potencialmente también pueden probarse enunciados adicionales. El conjunto de proposiciones demostrables crece monótonamente cuando se amplía el conjunto de fórmulas. Para nuestro ejemplo, esto significa que la ampliación de la base de conocimientos nunca conducirá a nuestro objetivo. Por lo tanto, modificamos KB reemplazando la declaración obviamente falsa "(todas) las aves pueden volar" con la declaración más exacta "(todas) las aves excepto los pingüinos pueden volar" y obtenemos como KB2 las siguientes cláusulas:

```
pingüino(tweety)
pingüino(x)  pájaro(x)
pájaro(x)  ¬pingüino(x)  mosca(x)
pingüino(x)  ¬mosca(x)
```

Ahora, aparentemente, el mundo vuelve a estar en orden. Podemos derivar $\neg \text{fly}(\text{tweety})$, pero no $\text{fly}(\text{tweety})$, porque para eso necesitaríamos $\neg \text{penguin}(x)$, que, sin embargo, no es derivable. Mientras solo haya pingüinos en este mundo, reina la paz. Todo pájaro normal, sin embargo, causa problemas de inmediato. Deseamos agregar el cuervo Abraxas (del libro alemán "La pequeña bruja") y obtener

```
cuervo(abraxas)
cuervo(x)  pájaro(x)
pingüino(tweety)
pingüino(x)  pájaro(x)
pájaro(x)  ¬pingüino(x)  mosca(x)
pingüino(x)  ¬mosca(x)
```

No podemos decir nada sobre los atributos de vuelo de Abraxas porque olvidamos formular que los cuervos no son pingüinos. Así extendemos KB3 a KB4:

```
cuervo(abraxas)
cuervo(x)  pájaro(x)
cuervo(x)  ¬pingüino(x)
pingüino(tweety)
pingüino(x)  pájaro(x)
pájaro(x)  ¬pingüino(x)  mosca(x)
pingüino(x)  ¬mosca(x)
```

El hecho de que los cuervos no son pingüinos, lo cual es evidente para los humanos, debe agregarse explícitamente aquí. Para la construcción de una base de conocimientos con los aproximadamente 9.800 tipos de aves en todo el mundo, se debe especificar para cada tipo de ave (excepto los pingüinos) que no es miembro de los pingüinos. Debemos proceder de manera análoga para todas las demás excepciones, como el avestruz.

Para cada objeto en la base de conocimiento, además de sus atributos, todos los deben enumerarse los atributos que no tiene.

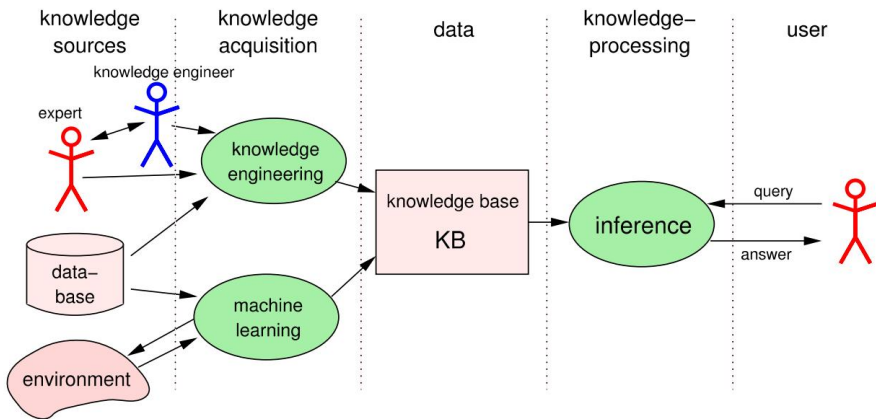


Fig. 1.6 Estructura de un sistema clásico de procesamiento de conocimiento

más fácil reemplazar la base de conocimiento de un sistema médico experto que programar un sistema completamente nuevo.

Mediante el desacoplamiento de la base de conocimiento de la inferencia, el conocimiento puede almacenarse declarativamente. En la base de conocimientos sólo hay una descripción de la conocimiento, que es independiente del sistema de inferencia en uso. Sin esta separación clara, el conocimiento y el procesamiento de los pasos de inferencia estarían entrelazados, y cualquier cambio en el conocimiento sería muy costoso.

El lenguaje formal como interfaz conveniente entre el hombre y la máquina se presta a la representación del conocimiento en la base de conocimiento. En los siguientes capítulos conoceremos toda una serie de tales lenguajes. Primero, en los Caps. 2 y 3 hay cálculo proposicional y lógica de predicados de primer orden (PL1). Pero también se presentan otros formalismos como la lógica probabilística, la lógica difusa o los árboles de decisión. Comenzamos con el cálculo proposicional y los sistemas de inferencia relacionados. Sobre la base de Eso, presentaremos la lógica de predicados, un lenguaje poderoso al que pueden acceder las máquinas y muy importante en la IA.

Como ejemplo de un sistema basado en el conocimiento a gran escala, queremos referirnos al agente de software "Watson". Desarrollado en IBM junto con varias universidades, Watson es un programa de respuesta a preguntas, que se puede alimentar con pistas dadas en natural idioma. Funciona sobre una base de conocimiento que comprende cuatro terabytes de almacenamiento en disco duro, incluido el texto completo de Wikipedia [FNA+09]. Watson fue desarrollado dentro Proyecto DeepQA de IBM que se caracteriza en [Dee11] de la siguiente manera:

El proyecto DeepQA de IBM da forma a un gran desafío en Ciencias de la Computación que tiene como objetivo ilustrar cómo la amplia y creciente accesibilidad del contenido del lenguaje natural y la integración y el avance del procesamiento del lenguaje natural, la recuperación de información, el aprendizaje automático, la representación y el razonamiento del conocimiento y la computación paralela masiva pueden impulsar la tecnología de respuesta automática de preguntas de dominio abierto a un nivel más alto. punto donde rivaliza clara y consistentemente con el mejor desempeño humano.

En el programa de concursos de la televisión estadounidense "Jeopardy!", en febrero de 2011, Watson derrotó a los dos campeones humanos Brad Rutter y Ken Jennings en un juego de dos juegos. partido de puntos combinados y ganó el precio de un millón de dólares. Uno de los par-

Una de sus fortalezas particulares fue su reacción muy rápida a las preguntas, con el resultado de que Watson a menudo tocaba el zumbador (usando un solenoide) más rápido que sus competidores humanos y luego podía dar la primera respuesta a la pregunta.

El alto rendimiento y los cortos tiempos de reacción de Watson se debieron a una implementación en 90 servidores IBM Power 750, cada uno de los cuales contiene 32 procesadores, lo que da como resultado 2880 procesadores paralelos.

1.5 Ejercicios

Ejercicio 1.1 Pruebe algunos de los chatterbots disponibles en Internet. Comience, por ejemplo, con www.hs-weingarten.de/~ertel/aibook en la colección de enlaces bajo Tur ingtest/Chatterbots, o en www.simonlaven.com o www.alicebot.org. Escriba una pregunta inicial y mida el tiempo que lleva, para cada uno de los diversos programas, hasta que sepa con certeza que no es un humano.

Ejercicio 1.2 En www.pandorabots.com encontrarás un servidor en el que puedes construir un chatterbot con el lenguaje de marcado AIML con bastante facilidad. Dependiendo de su nivel de interés, desarrolle un chatterbot simple o complejo, o cambie uno existente.

Ejercicio 1.3 Explique las razones de la inadecuación de la prueba de Turing como definición de "inteligencia artificial" en la IA práctica.

Ejercicio 1.4 Muchos procesos de inferencia, procesos de aprendizaje, etc. conocidos son NP-completo o incluso indecidible. ¿Qué significa esto para la IA?

Ejercicio 1.5 (a)

¿Por qué un agente determinista con memoria no es una función del conjunto de todas las entradas al conjunto de todas las salidas, en el sentido matemático? (b) ¿Cómo se puede cambiar el agente con la memoria, o modelarlo, de tal manera que se convierta en equivalente a una función pero no pierde su memoria?

Ejercicio 1.6 Sea un agente con memoria que pueda moverse dentro de un plano. De sus sensores, recibe ticks de reloj de un intervalo regular en su posición exacta (x, y) en coordenadas cartesianas. (a) Dé una fórmula con la que el agente

pueda calcular su velocidad a partir de la corriente tiempo t y la medida anterior de $t - t$.

(b) ¿Cómo debe cambiarse el agente para que también pueda calcular su aceleración? Proporcione una fórmula aquí también.

Ejercicio 1.7

- (a) Determine para ambos agentes en el Ejemplo 1.1 en la página 11 los costos creados por los errores y compare los resultados. Supongamos aquí que tener que eliminar manualmente un correo electrónico no deseado cuesta un centavo y recuperar un correo electrónico eliminado, o la pérdida de un correo electrónico, cuesta un dólar.
- (b) Determine para ambos agentes la ganancia creada por las clasificaciones correctas y compare los resultados. Suponga que por cada correo electrónico deseado reconocido, se acumula una ganancia de un dólar y por cada correo electrónico no deseado eliminado correctamente, una ganancia de un centavo.

Lógica proposicional

2

En la lógica proposicional, como sugiere su nombre, las proposiciones están conectadas por operadores lógicos. El enunciado “la calle está mojada” es una proposición, como lo es “está lloviendo”. Estas dos proposiciones se pueden conectar para formar la nueva proposición

si está lloviendo la calle está mojada.

Escrito más formalmente

está lloviendo la calle está mojada.

Esta notación tiene la ventaja de que las proposiciones elementales aparecen de nuevo en forma inalterada. Para que podamos trabajar con lógica proposicional con precisión, comenzaremos con una definición del conjunto de todas las fórmulas de lógica proposicional.

2.1 Sintaxis

Definición 2.1 Sea $Op = \{\neg, \vee, \wedge, \rightarrow, \leftrightarrow, (\cdot,)\}$ el conjunto de operadores lógicos y Σ un conjunto de símbolos. Los conjuntos Op , Σ y $\{t, f\}$ son disjuntos por parejas. Σ se llama firma y sus elementos son las variables de la proposición. El conjunto de fórmulas de lógica proposicional ahora se define recursivamente:

- t y f son fórmulas (atómicas).
- Todas las variables de proposición, es decir, todos los elementos de Σ , son fórmulas (atómicas) las.
- Si A y B son fórmulas, entonces $\neg A$, $(A) \vee B$, $A \wedge B$, $A \rightarrow B$, $A \leftrightarrow B$ también son fórmulas.

Esta elegante definición recursiva del conjunto de todas las fórmulas nos permite generar infinitas fórmulas. Por ejemplo, dado $\Sigma = \{A, B, C\}$,

$A \vee B$, $A \wedge B$, $C \vee A$, $A \wedge C$, $B \vee A$, $(\neg A \vee B) \wedge (\neg C \vee A)$

son fórmulas. $((A) \vee B)$ también es una fórmula sintácticamente correcta.

Definición 2.2 Leemos los símbolos y operadores de la siguiente manera:

t:	"verdadero"	
f:	"falso"	
$\neg A$:	"no A"	(negación)
A B:	"A y B"	(conjunción)
A B:	"A o B"	(disyunción)
A B:	"si A entonces B"	(implicación (también llamada implicación material))
A B:	"A si y solo si B"	(equivalencia)

Las fórmulas así definidas son hasta ahora construcciones puramente sintácticas sin significado. Todavía nos falta la semántica.

2.2 Semántica

En lógica proposicional hay dos valores de verdad: t para "verdadero" y f para "falso". Nosotros Comenzamos con un ejemplo y nos preguntamos si la fórmula $A \rightarrow B$ es verdadera. El La respuesta es: depende de si las variables A y B son verdaderas. Por ejemplo, si A significa "Hoy está lloviendo" y B para "Hoy hace frío" y ambas son verdaderas, entonces $A \rightarrow B$ es verdadera. Sin embargo, si B representa "Hoy hace calor" (y esto es falso), entonces $A \rightarrow B$ es falso.

Obviamente, debemos asignar valores de verdad que reflejen el estado del mundo para proponer variables de posición. Por lo tanto definimos

Definición 2.3 Una aplicación $I : \Sigma \rightarrow \{w, f\}$, que asigna un valor de verdad a toda proposición variable, se llama interpretación.

Debido a que cada variable proposicional puede tomar dos valores de verdad, cada fórmula lógica proposicional con n variables diferentes tiene 2^n interpretaciones diferentes. Nosotros definamos los valores de verdad para las operaciones básicas mostrando todas las interpretaciones posibles en una tabla de verdad (vea la Tabla 2.1 en la página 17).

La fórmula vacía es verdadera para todas las interpretaciones. Para determinar la verdad valor para fórmulas complejas, también debemos definir el orden de las operaciones para fórmulas lógicas operadores. Si las expresiones están entre paréntesis, se evalúa el término entre paréntesis

Tabla 2.1 Definición de la
operadores lógicos por verdad
mesa

AB (A) $\neg$ AA	BA	BA	BA	B
pies				t
t pies ff				t
pies pies		F		t
fff t		F	F	t

primero. Para fórmulas sin paréntesis, las prioridades se ordenan de la siguiente manera, comenzando con la unión más fuerte: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

Diferenciar claramente entre la equivalencia de fórmulas y el equiv sintáctico
alencia, definimos

Definición 2.4 Dos fórmulas F y G se llaman semánticamente equivalentes si toman el mismo valor de verdad para todas las interpretaciones. Escribimos $F \equiv G$.

La equivalencia semántica sirve sobre todo para poder utilizar el metalenguaje, que es, lenguaje natural, para hablar del lenguaje objeto, es decir, la lógica. La declaración “ $A \equiv B$ ” transmite que las dos fórmulas A y B son semánticamente equivalentes. El enunciado “ $A \rightarrow B$ ” por otro lado es un objeto sintáctico del lenguaje formal de la lógica proposicional.

Según cuántas interpretaciones en las que una fórmula es verdadera, podemos dividir fórmulas en las siguientes clases:

Definición 2.5 Una fórmula se llama

- Satisfecho si es cierto para al menos una interpretación.
- Lógicamente válido o simplemente válido si es verdadero para todas las interpretaciones. Las fórmulas verdaderas también se llaman tautologías.
- Insatisfactorio si no es cierto para alguna interpretación.

Toda interpretación que satisface una fórmula se llama modelo de la fórmula.

Claramente, la negación de toda fórmula generalmente válida es insatisfactoria. el negación de una fórmula F satisfactoria, pero generalmente no válida, es satisfactoria.

Ahora podemos crear tablas de verdad para fórmulas complejas para determinar su valores de verdad. Ponemos esto en acción inmediatamente usando equivalencias de fórmulas que son importantes en la práctica.

Teorema 2.1 Las operaciones $+$, $\cdot$ son conmutativas y asociativas, y las

Las siguientes equivalencias son generalmente válidas:

	$\neg A$	B	A	B		(implicación)
	A	B	$\neg B$	$\neg A$		(contraposición)
$(A \quad B)$	$(B \quad A)$	$(A \quad B)$	$\neg(A \quad B)$	$\neg A$		(equivalencia)
	$\neg A$	$\neg B$	$\neg(A \quad B)$	$\neg A$		(Ley de De Morgan)
	$\neg B$					
	A	$(B \quad C)$	$(A \quad B)$	$(A \quad C)$		(ley distributiva)
	A	$(B \quad C)$	$(A \quad B)$	$(A \quad C)$		
	A	$\neg A$	w			(tautología)
	A	$\neg A$	f			(contradicción)
	A	f	A			
	A	w	w			
	A	f	f			
	A	w	A			

Prueba Para mostrar la primera equivalencia, calculamos la tabla de verdad para $\neg A \vee B$ y $A \rightarrow B$ y vea que los valores de verdad para ambas fórmulas son los mismos para todas las interpretaciones. Por lo tanto, las fórmulas son equivalentes y, por lo tanto, todos los valores de la última columna son "t"

AB	$\neg A$	$\neg A$	BA	B	$(\neg A \vee B)$	$(A \wedge B)$
t	t	f	f	f	t	f
t	t	f	f	t	t	t
t	f	t	t	f	t	f
t	f	t	t	t	t	t
f	t	t	f	f	t	f
f	t	t	f	t	t	f
f	f	t	t	f	t	f
f	f	t	t	t	t	f

Las demostraciones para las demás equivalencias son similares y se recomiendan como ejercicios para el lector (Ejercicio 2.2 de la página 29).

2.3 Sistemas de prueba

En IA estamos interesados en tomar el conocimiento existente y de ahí derivar nuevos conocimientos o responder preguntas. En lógica proposicional esto significa mostrar que una base de conocimiento KB, es decir, una fórmula lógica proposicional (posiblemente extensa) sigue una fórmula Q1 . Por lo tanto, primero definimos el término "vinculación".

1Aquí Q significa consulta.

Definición 2.6 Una fórmula KB implica una fórmula Q (o Q se sigue de KB) si todo modelo de KB es también un modelo de Q. Escribimos $KB \models Q$

En otras palabras, en cada interpretación en la que KB es verdadera, Q también lo es. Más sucintamente, siempre que KB es verdadero, Q también lo es. Como para el concepto de implicación se introducen interpretaciones de variables, estamos ante un concepto semántico.

Toda fórmula que no es válida elige, por así decirlo, un subconjunto del conjunto de todas las interpretaciones como su modelo. Tautologías como $A \rightarrow A$, por ejemplo, no restringen el número de interpretaciones satisfactorias porque su proposición está vacía. Por lo tanto, la fórmula vacía es verdadera en todas las interpretaciones. Para toda tautología T entonces $\models T$ Intuitivamente esto significa que las tautologías son siempre verdaderas, sin restricción de las interpretaciones por una fórmula. Para abreviar escribimos $\models$ Ahora mostramos una conexión importante entre el concepto semántico de implicación y la implicación sintáctica.

Teorema 2.2 (Teorema de las deducciones)

$A \models B$ si y solo si $A \vdash B$.

Prueba Observe la tabla de verdad para la implicación:

A	B
t	t
t	f
f	t
f	f

Una implicación arbitraria $A \rightarrow B$ es claramente siempre verdadera excepto con la interpretación $A \rightarrow t, B \rightarrow f$. Suponga que $A \models B$ aguanta. Esto significa que por cada interpretación que hace que A sea verdadera, B también lo es. La segunda fila crítica de la tabla de verdad ni siquiera se aplica en ese caso. Por lo tanto, $A \rightarrow B$ es verdadero, lo que significa que $A \rightarrow B$ es una tautología. Así se ha mostrado una dirección del enunciado.

Ahora suponga que $A \vdash B$ se cumple. Por lo tanto, la segunda fila crítica de la tabla de verdad es también bloqueado. Todo modelo de A es entonces también un modelo de B. Entonces $A \models B$ aguanta.

Si queremos demostrar que KB implica Q, también podemos demostrar mediante el método de la tabla de verdad que $KB \models Q$ es una tautología. Así tenemos nuestro primer sistema de prueba para la lógica proposicional, que se automatiza fácilmente. La desventaja de este método es el tiempo de cálculo muy largo en el peor de los casos. Específicamente, en el peor de los casos con n variables de proposición, para todas las 2^n interpretaciones de las variables se debe evaluar la fórmula $KB \models Q$. El tiempo de cálculo crece por tanto exponencialmente

con el número de variables. Por lo tanto, este proceso es inutilizable para grandes recuentos de variables, al menos en el peor de los casos.

Si una fórmula KB implica una fórmula Q , entonces por el teorema de deducción $KB \vdash Q$ es una tautología. Por tanto, la negación $\neg(KB \vdash Q)$ es insatisfactoria. Tenemos

$$\neg(KB \vdash Q) \equiv \neg(\neg KB \vdash Q) \equiv KB \vdash \neg Q.$$

Por lo tanto $KB \vdash \neg Q$ también es satisficible. Formulamos esta consecuencia simple pero importante del teorema de deducción como un teorema.

Teorema 2.3 (Demostración por contradicción) $KB \vdash Q$ si y sólo si $KB \vdash \neg Q$ es insatisficible.

Para mostrar que la consulta Q se deriva de la base de conocimiento KB , también podemos agregar la consulta negada $\neg Q$ a la base de conocimiento y derivar una contradicción. Debido a la equivalencia $A \vdash \neg A$ f del Teorema 2.1 en la página 18, sabemos que una contradicción es insatisfecha. Por lo tanto, Q ha sido probada. Este procedimiento, que se usa con frecuencia en matemáticas, también se usa en varios cálculos de prueba automática, como el cálculo de resolución y en el procesamiento de programas PROLOG.

Una forma de evitar tener que probar todas las interpretaciones con el método de la tabla de verdad es la manipulación sintáctica de las fórmulas KB y Q mediante la aplicación de reglas de inferencia con el objetivo de simplificarlas en gran medida, de modo que al final podamos ver instantáneamente que $KB \vdash P$. A este proceso sintáctico lo llamamos derivación y escribimos $KB \vdash P$. Tales sistemas de prueba sintáctica se denominan cálculos. Para asegurar que un cálculo no genere errores, definimos dos propiedades fundamentales de los cálculos.

Definición 2.7 Un cálculo se llama sonido si toda proposición derivada se sigue semánticamente. Es decir, si se cumple para las fórmulas KB y Q que

$$\text{si } KB \vdash Q \text{ entonces } KB \models Q$$

Un cálculo se dice completo si se pueden derivar todas las consecuencias semánticas. Es decir, para las fórmulas KB y Q se cumple lo siguiente:

$$\text{si } KB \models Q \text{ entonces } KB \vdash Q.$$

La solidez de un cálculo asegura que todas las fórmulas derivadas sean de hecho consecuencias semánticas de la base de conocimientos. El cálculo no produce ninguna "falsa consecuencia". La completitud de un cálculo, por otro lado, asegura que el cálculo no pase por alto nada. Un cálculo completo siempre encuentra una demostración si la fórmula a demostrar se deriva de la base de conocimientos. Si un cálculo es sólido

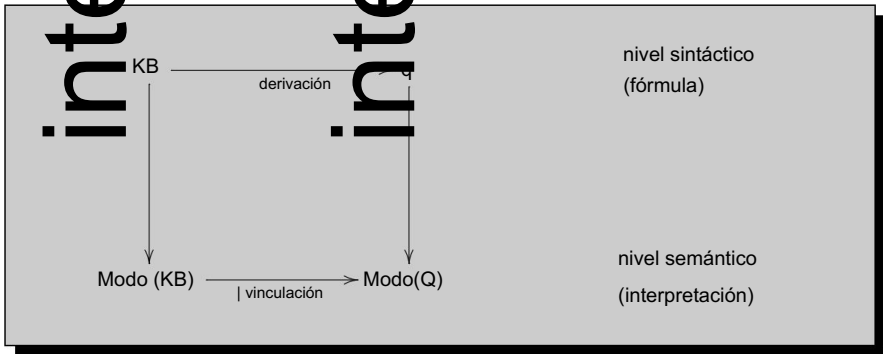


Fig. 2.1 Derivación sintáctica y vinculación semántica. $\text{Mod}(X)$ representa el conjunto de modelos de una fórmula X

y completa, entonces la derivación sintáctica y la implicación semántica son dos relaciones equivalentes (ver Fig. 2.1).

Para mantener los sistemas de prueba automáticos lo más simples posible, estos generalmente se hacen operar sobre fórmulas en forma conjuntiva normal.

Definición 2.8 Una fórmula está en forma normal conjuntiva (CNF) si y solo si consiste en una conjunción

$$K_1 \quad K_2 \quad \dots \quad K_m$$

de cláusulas. Una cláusula K_i consiste en una disyunción

$$(L_{i1} \quad L_{i2} \quad \dots \quad L_{ini})$$

de literales. Finalmente, un literal es una variable (literal positivo) o una variable negada (literal negativo).

La fórmula $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ está en forma normal conjuntiva. La forma normal conjuntiva no pone restricción al conjunto de fórmulas porque:

Teorema 2.4 Toda fórmula lógica proposicional puede transformarse en una forma normal conjuntiva equivalente.

Ejemplo 2.1 Ponemos $A \vee B \vee C \vee D$ en forma normal conjuntiva usando las equivalencias del Teorema 2.1 en la página 18:

$$\begin{aligned} A \vee B \vee C \vee D &\equiv \\ \neg(A \vee B) \wedge \neg(C \vee D) & \text{ (implicación)} \equiv (\neg A \wedge \neg B) \wedge (\neg C \wedge \neg D) \text{ (de Morgan)} \equiv \\ (\neg A \wedge \neg C) \vee (\neg A \wedge \neg D) \vee & (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D) \text{ (ley distributiva)} \equiv ((\neg A \wedge \neg C) \vee (\neg A \wedge \neg D) \vee \\ & (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)) \text{ (ley distributiva)} \equiv (\neg A \wedge \neg C) \vee (\neg A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D) \text{ (ley asociativa)} \end{aligned}$$

Ahora solo nos falta un cálculo para la prueba sintáctica de fórmulas de lógica proposicional. Comenzamos con el modus ponens, una regla de inferencia simple e intuitiva que, a partir de la validez de A y $A \rightarrow B$, permite la derivación de B . Escribimos esto formalmente como

$$\frac{A, A \rightarrow B}{B}.$$

Esta notación significa que podemos derivar la(s) fórmula(s) debajo de la línea a partir de las fórmulas separadas por comas arriba de la línea. Modus ponens como regla por sí mismo, mientras que el sonido, no es completo. Si agregamos reglas adicionales, podemos crear un cálculo completo que, sin embargo, no deseamos considerar aquí. En su lugar, investigaremos la regla de resolución.

$$\frac{A \vee B, \neg B \vee C}{A \vee C} \quad (2.1)$$

como alternativa. La cláusula derivada se llama resolvente. Mediante una transformación simple obtenemos la forma equivalente

$$\frac{A \vee B, B \vee C}{A \vee C}.$$

Si ponemos A a f , vemos que la regla de resolución es una generalización del modus ponens. La regla de resolución es igualmente útil si falta C o si faltan A y C . En este último caso la cláusula vacía puede derivarse de la contradicción $B \vee \neg B$ (Ejercicio 2.7 de la página 30).

2.4 Resolución

Ahora generalizamos nuevamente la regla de resolución al permitir cláusulas con un número arbitrario de literales. Con los literales $A_1, \dots, A_m, B, C_1, \dots, C_n$ la regla general de resolución queda como

$$\frac{(A_1 \vee \dots \vee A_m \vee B), (\neg B \vee C_1 \vee \dots \vee C_n)}{(A_1 \vee \dots \vee A_m \vee C_1 \vee \dots \vee C_n)} \quad (2.2)$$

Llamamos a los literales B y $\neg B$ complementarios. La regla de resolución elimina un par de literales complementarios de las dos cláusulas y combina el resto de los literales en una nueva cláusula.

Para probar que de una base de conocimiento KB se sigue una consulta Q , realizamos una prueba por contradicción. Siguiendo el Teorema 2.3 de la página 20 debemos demostrar que se puede derivar una contradicción de $KB \cup \neg Q$. En las fórmulas en forma normal conjuntiva aparece una contradicción en forma de dos cláusulas (A) y $(\neg A)$, que llevan a la cláusula vacía como su resolutiva. El siguiente teorema nos asegura que este proceso realmente funciona como se desea.

Para que el cálculo esté completo, necesitamos una pequeña adición, como se muestra en el siguiente ejemplo. Deje que la fórmula $(A \vee A)$ se dé como nuestra base de conocimiento. Para mostrar por la regla de resolución que de allí podemos derivar $(A \vee A)$, debemos mostrar que la cláusula vacía se puede derivar de $(A \vee A) \wedge (\neg A \vee \neg A)$. Solo con la regla de resolución, esto es imposible. Con la factorización, que permite eliminar copias de literales de cláusulas, se elimina este problema. En el ejemplo, una doble aplicación de factorización conduce a $(A) \wedge (\neg A)$, y un paso de resolución a la cláusula vacía.

Teorema 2.5 El cálculo de resolución para la prueba de insatisfacción de fórmulas en forma normal conjuntiva es correcto y completo.

Porque el trabajo del cálculo de resolución es derivar una contradicción de $KB \cup \neg Q$, es muy importante que la base de conocimiento KB sea consistente:

Definición 2.9 Una fórmula KB se llama consistente si es imposible derivar de ella una contradicción, es decir, una fórmula de la forma $\phi \wedge \neg \phi$.

De lo contrario, se puede derivar cualquier cosa de KB (vea el Ejercicio 2.8 en la página 30). Este es cierto no sólo para la resolución, sino también para muchos otros cálculos.

De los cálculos por deducción automática, la resolución juega un papel excepcional. Por lo tanto, deseamos trabajar un poco más de cerca con él. A diferencia de otros cálculos, la resolución tiene solo dos reglas de inferencia y trabaja con fórmulas en forma normal conjuntiva. Esto hace que su implementación sea más sencilla. Una ventaja adicional frente a muchos cálculos radica en su reducción del número de posibilidades para la aplicación de reglas de inferencia en cada paso de la demostración, por lo que se reduce el espacio de búsqueda y el tiempo de cálculo.

Como ejemplo, comenzamos con un rompecabezas de lógica simple que permite los pasos importantes de una prueba de resolución a mostrar.

Ejemplo 2.2 Acertijo de lógica número 7, titulado Una encantadora familia inglesa, del libro alemán [Ber89] dice (traducido al inglés):

A pesar de estudiar inglés durante siete largos años con un éxito brillante, debo admitir que cuando escucho a los ingleses hablar en inglés me quedo totalmente perplejo. Recientemente, movido por sentimientos nobles, recogí a tres autoestopistas, un padre, una madre y una hija, de quienes rápidamente me di cuenta que eran ingleses y solo hablaban inglés. En cada una de las oraciones que siguen vacilé entre dos posibles interpretaciones. Me dijeron lo siguiente (el segundo significado posible está entre paréntesis): El padre: "Nos vamos a España (somos de Newcastle)". La madre: "No vamos a España y somos de Newcastle (paramos en París y no vamos a España)". La hija: "No somos de Newcastle (paramos en París)". ¿Qué hay de esta encantadora familia inglesa?

Para resolver este tipo de problema se procede en tres pasos: formalización, transformación a forma normal y demostración. En muchos casos, la formalización es, con mucho, el paso más difícil porque es fácil cometer errores u olvidar pequeños detalles. (Por lo tanto, el ejercicio práctico es muy importante. Véanse los Ejercicios 2.9–2.11.)

Aquí usamos las variables S para "Nos vamos a España", N para "Somos de Newcastle", y P por "Nos detuvimos en París" y obtienen como formalización de las tres proposiciones de padre, madre e hija

$$(S \vee N) \wedge [(\neg S \vee N) \wedge (P \vee \neg S)] \wedge (\neg N \vee P).$$

Factorizar $\neg S$ en la subfórmula del medio lleva la fórmula a CNF en un solo paso. Numerar las cláusulas con índices subíndices produce

$$KB \equiv (S \vee N) \wedge (\neg S) \wedge (P \vee N) \wedge (\neg N \vee P).$$

Ahora comenzamos la prueba de resolución, al principio aún sin una consulta Q. Una expresión de la forma "Res(m, n): cláusula_k" significa que la cláusula se obtiene mediante la resolución de la cláusula m con la cláusula n y se numera como k.

$$\text{Res}(1, 2): (N) \wedge 5$$

$$\text{Res}(3, 4): (P) \wedge 6$$

$$\text{Res}(1, 4): (S \vee P) \wedge 7$$

Podríamos haber derivado la cláusula P también de Res(4, 5) o Res(2, 7). Cada paso de resolución adicional conduciría a la derivación de cláusulas que ya están disponibles. Debido a que no permite la derivación de la cláusula vacía, se ha demostrado que la base de conocimiento no es contradictoria. Hasta ahora hemos derivado N y P. Para mostrar que $\neg S$ se cumple, agregamos la cláusula (S)8 al conjunto de cláusulas como una consulta negada. Con el paso de resolución

$$\text{Res}(2, 8): () \wedge 9$$

la prueba es completa. Por lo tanto, $\neg S \vee N \vee P$ se cumple. La "encantadora familia inglesa" evidentemente viene de Newcastle, parada en París, pero no va a España.

Ejemplo 2.3 El rompecabezas lógico número 28 de [Ber89], titulado The High Jump, dice

Tres niñas practican salto de altura para su examen final de educación física. La barra está puesta en 1,20 metros. "Apuesto", le dice la primera chica a la segunda, "que lo lograré si, y sólo si, tú no lo haces". Si la segunda chica le dijera lo mismo a la tercera, quien a su vez le dijo lo mismo a la primera, ¿sería posible que las tres ganaran sus apuestas?

Mostramos a través de la prueba por resolución que no los tres pueden ganar sus apuestas.

Formalización:

El salto de la primera niña tiene éxito: A, el	Apuesta de la primera chica: (A
salto de la segunda niña tiene éxito: B, el salto	$\neg B$), apuesta de la segunda chica: (B
de la tercera niña tiene éxito: C.	$\neg C$), apuesta de la tercera chica: (C $\neg A$).

Reclamo: los tres no pueden ganar todos sus apuestas:

$$Q \equiv \neg((A \rightarrow B) \wedge (B \rightarrow C) \wedge (C \rightarrow A))$$

Ahora debe demostrarse mediante resolución que $\neg Q$ es insatisfacible.

Transformación en CNF: Apuesta de la primera chica:

$$(A \rightarrow B) \equiv (A \rightarrow B) \wedge (\neg B \rightarrow A) \equiv (\neg A \vee B) \wedge (A \vee \neg B)$$

Las apuestas de las otras dos chicas sufren transformaciones análogas y obtenemos el reclamo negado.

$$\neg Q \equiv (\neg A \vee \neg B) \wedge (A \vee B) \wedge (\neg B \vee \neg C) \wedge (B \vee C) \wedge (\neg C \vee \neg A) \wedge (C \vee A) \wedge 6.$$

De ahí derivamos la cláusula vacía usando resolución:

$$\text{Res}(1, 6): (C \vee \neg B) \quad 7$$

$$\text{Res}(4, 7): (C) \quad 8$$

$$\text{Res}(2, 5): (B \vee \neg C) \quad 9$$

$$\text{Res}(3, 9): (\neg C) \quad 10$$

$$\text{Res}(8, 10): ()$$

Por lo tanto, la afirmación ha sido probada.

2.5 Cláusulas Horn

Una cláusula en forma normal conjuntiva contiene literales positivos y negativos y se puede representar en la forma

$$(\neg A_1 \vee \dots \vee \neg A_m \vee B_1 \vee \dots \vee B_n)$$

con las variables $A_1, \dots, A_m$ y $B_1, \dots, B_n$. Esta cláusula se puede transformar en dos simples pasos en la forma equivalente

$$A_1 \quad \dots \quad A_m \quad B_1 \quad \dots \quad B_n.$$

Esta implicación contiene la premisa, una conjunción de variables y la conclusión, una disyunción de variables. Por ejemplo, "Si hace buen tiempo y hay nieve en el suelo, iré a esquiar o trabajaré". es una proposición de esta forma. El receptor de este mensaje sabe con certeza que el remitente no va a nadar.

Una declaración significativamente más clara sería "Si hace buen tiempo y hay nieve en el suelo, iré a esquiar". El receptor ahora sabe definitivamente. Así llamamos cláusulas con a lo sumo un literal definido positivo cláusulas. Estas cláusulas tienen la ventaja de que solo permiten una conclusión y, por lo tanto, son claramente más simples de interpretar. Muchas relaciones pueden describirse mediante cláusulas de este tipo. Por lo tanto, definimos

Definición 2.10 Cláusulas con como máximo un literal positivo de la forma

$$(\neg A_1 \quad \dots \quad \neg A_m \quad B) \text{ o } (\neg A_1 \quad \dots \quad \neg A_m) \text{ o } B$$

o equivalente)

$$A_1 \quad \dots \quad A_m \quad B \text{ o } A_1 \quad \dots \quad A_m \quad \text{f o } B.$$

se denominan cláusulas de Horn (en honor a su inventor). Una cláusula con un solo literal positivo es un hecho. En las cláusulas con un literal negativo y uno positivo, el literal positivo se llama encabezado .

Para comprender mejor la representación de las cláusulas de Horn, el lector puede derivarlas de las definiciones de las equivalencias que hemos estado usando actualmente (Ejercicio 2.12 en la página 30).

Las cláusulas de cuerno son más fáciles de manejar no solo en la vida diaria, sino también en el razonamiento formal, como podemos ver en el siguiente ejemplo. Deje que la base de conocimiento consista en las siguientes cláusulas (el " " que vincula las cláusulas se omite aquí y en el texto que sigue):

(buen_tiempo)1

(nevada)2

(nevada nieve)3

(buen_tiempo nieve esquí)4

Si ahora queremos saber si el esquí aguanta, esto se puede deducir fácilmente. Un modus ponens ligeramente generalizado es suficiente aquí como regla de inferencia:

$$\frac{A_1 \quad \dots \quad A_m, A_1 \quad \dots \quad A_m \quad B}{B} .$$

La prueba de "esquí" tiene la siguiente forma (MP(i1,...,ik) representa la aplicación del modus ponens en las cláusulas i1 a ik :

MP(2, 3): (nieve)⁵ MP(1,
5, 4): (esquí)⁶.

Con modus ponens obtenemos un cálculo completo para fórmulas que consisten en cláusulas Horn de lógica proposicional. Sin embargo, en el caso de grandes bases de conocimiento, el modus ponens puede generar muchas fórmulas innecesarias si se comienza con las cláusulas incorrectas. Por lo tanto, en muchos casos es mejor usar un cálculo que comience con la consulta y trabaje hacia atrás hasta llegar a los hechos. Dichos sistemas se denominan encadenamiento hacia atrás, en contraste con los sistemas de encadenamiento hacia adelante, que comienzan con hechos y finalmente derivan la consulta, como en el ejemplo anterior con el modus ponens.

Para el encadenamiento hacia atrás de las cláusulas Horn, se utiliza la resolución SLD. SLD significa "Resolución lineal basada en reglas de selección para cláusulas definidas". En el ejemplo anterior, aumentado por la consulta negada (esquí f)

(buen_tiempo)¹
(nevada)²
(nevada nieve)³
(buen_tiempo nieve esquí)⁴
(esquí f)⁵

llevamos a cabo la resolución de SLD comenzando con los pasos de resolución que siguen a partir de esta cláusula

Res(5, 4): (buen_tiempo nieve f)⁶
Res(6, 1): (nieve f)⁷
Res(7, 3): (nevada f)⁸
Res(8, 2): ()

y derivar una contradicción con la cláusula vacía. Aquí podemos ver fácilmente "resolución lineal", lo que significa que siempre se realiza un procesamiento adicional en la cláusula derivada actualmente. Esto conduce a una gran reducción del espacio de búsqueda. Además, los literales de la cláusula actual siempre se procesan en un orden fijo (por ejemplo, de derecha a izquierda) ("Regla de selección impulsada"). Los literales de la cláusula actual se llaman

subobjetivo Los literales de la consulta negada son los objetivos. La regla de inferencia para un paso dice

$$\frac{A_1 \quad \dots \quad A_m \quad B_1, B_1 \quad B_2 \quad \dots \quad B_n \quad f}{A_1 \quad \dots \quad A_m \quad B_2 \quad \dots \quad B_n \quad f}$$

Antes de la aplicación de la regla de inferencia, $B_1, B_2, \dots, B_n$, las submetas actuales, deben probarse. Después de la aplicación, B_1 se reemplaza por el nuevo subobjetivo $A_1 \quad \dots \quad A_m$. Para demostrar que B_1 es verdadero, ahora debemos demostrar que $A_1 \quad \dots \quad A_m$ son verdaderos. Este proceso continúa hasta que la lista de subobjetivos de las cláusulas actuales (la llamada pila de objetivos) está vacía. Con eso, se ha encontrado una contradicción. Si, para un subobjetivo $\neg B_i$, no hay una cláusula con el literal complementario B_i como cabeza de cláusula, la prueba termina y no se puede encontrar ninguna contradicción. La consulta es, por lo tanto, indemostrable.

La resolución SLD juega un papel importante en la práctica porque los programas en el lenguaje de programación lógica PROLOG consisten en cláusulas Horn de lógica de predicados, y su procesamiento se logra mediante la resolución SLD (vea el Ejercicio 2.13 en la página 30, o el [Capítulo 5](#)).

2.6 Computabilidad y Complejidad

El método de la tabla de verdad, como el sistema de prueba semántica más simple para la lógica proposicional, representa un algoritmo que puede determinar cada modelo de cualquier fórmula en un tiempo finito. Así, los conjuntos de fórmulas insatisfactorias, satisfactorias y válidas son decidibles. El tiempo de cálculo del método de la tabla de verdad para la satisfacibilidad crece exponencialmente en el peor de los casos con el número n de variables porque la tabla de verdad 2^n tiene filas. Una optimización, el método de los árboles semánticos, evita mirar variables que no ocurren en las cláusulas y, por lo tanto, ahorra tiempo de cálculo en muchos casos, pero en el peor de los casos es igualmente exponencial.

En resolución, en el peor de los casos el número de cláusulas derivadas crece exponencialmente con el número de cláusulas iniciales. Para decidir entre los dos procesos, podemos usar la regla general de que en el caso de muchas cláusulas con pocas variables, es preferible el método de la tabla de verdad, y en el caso de pocas cláusulas con muchas variables, la resolución probablemente terminará más rápido.

La pregunta sigue siendo: ¿puede la prueba en lógica proposicional ir más rápido? ¿Hay mejores algoritmos? La respuesta: probablemente no. Después de todo, S. Cook, el fundador de la teoría de la complejidad, ha demostrado que el problema 3-SAT es NP-completo. 3-SAT es el conjunto de todas las fórmulas CNF cuyas cláusulas tienen exactamente tres literales. Por lo tanto, está claro que probablemente (módulo al problema P/NP) no haya un algoritmo polinomial para 3-SAT y, por lo tanto, probablemente tampoco uno general. Sin embargo, para las cláusulas de Horn, existe un algoritmo en el que el tiempo de cálculo para probar la satisfacibilidad crece solo linealmente a medida que aumenta el número de literales en la fórmula.

2.7 Aplicaciones y limitaciones

Los probadores de teoremas para la lógica proposicional son parte del conjunto de herramientas diarias del desarrollador en tecnología digital. Por ejemplo, la verificación de circuitos digitales y la generación de patrones de prueba para probar microprocesadores en fabricación son algunas de estas tareas. Los sistemas de prueba especiales que funcionan con diagramas de decisión binarios (BDD) () también se emplean como estructura de datos para procesar fórmulas de lógica proposicional.

En AI, la lógica proposicional se emplea en aplicaciones simples. Por ejemplo, los sistemas expertos simples ciertamente pueden funcionar con lógica proposicional. Sin embargo, todas las variables deben ser discretas, con solo unos pocos valores, y no puede haber ninguna relación cruzada entre las variables. Las conexiones lógicas complejas se pueden expresar de manera mucho más elegante utilizando la lógica de predicados.

La lógica probabilística es una combinación muy interesante y actual de lógica proposicional y computación probabilística que permite modelar el conocimiento incierto.

Se trata a fondo en el cap. 7. La lógica difusa, que permite infinitos valores de verdad, también se analiza en ese capítulo.

2.8 Ejercicios

Ejercicio 2.1 Dar una gramática de forma Backus-Naur para la sintaxis de proposiciones lógica.

Ejercicio 2.2 Muestre que las siguientes fórmulas son tautologías: (a) $\neg(A$

B) $\neg A \quad \neg B$ (b) $A \quad B \quad \neg B$

$\neg A$ (c) $((A \quad B) \quad (B \quad A))$

$(A \quad B)$ (d) $(A \quad B) \quad (\neg B \quad C) \quad (A \quad C)$

Ejercicio 2.3 Transforma las siguientes fórmulas a la forma normal conjuntiva: (a) $A \quad B$ (b) A

$B \quad A$

B (c) $A \quad (A \quad B) \quad B$

Ejercicio 2.4 Verifique las siguientes afirmaciones para ver si son satisfactorias o válidas.

(a) $(\text{play\_lottery} \quad \text{six\_right}) \quad \text{ganador}$ (b)

$(\text{play\_lottery} \quad \text{six\_right} \quad (\text{six\_right} \quad \text{win})) \quad \text{win}$ (c) $\neg(\neg \text{gas\_in\_tank}$

$(\text{gas\_in\_tank} \quad \neg \text{car\_starts}) \quad \neg \text{car\_starts})$

Ejercicio 2.5 Usando el lenguaje de programación de su elección, programe un probador de teoremas para lógica proposicional usando el método de la tabla de verdad para fórmulas en forma normal conjuntiva. Para evitar una costosa verificación de sintaxis de las fórmulas, puede representar las cláusulas como listas o conjuntos de literales y las fórmulas como listas o conjuntos de cláusulas. El programa debe indicar si la fórmula es insatisfactoria, satisfactoria o verdadera, y generar el número de interpretaciones y modelos diferentes.

Ejercicio 2.6 (a)

Demuestre que modus ponens es una regla de inferencia válida mostrando que $A \rightarrow B \mid B$. (b)

Demuestre que la

regla de resolución (2.1) es una regla de inferencia válida.

Ejercicio 2.7 Demostrar por aplicación de la regla de resolución que, en conjunción normal forma, la cláusula vacía es equivalente a la declaración falsa.

Ejercicio 2.8 Muestre que, con resolución, uno puede "derivar" cualquier cláusula arbitraria de una base de conocimiento que contiene una contradicción.

Ejercicio 2.9 Formalice las siguientes funciones lógicas con los operadores lógicos y demuestre que su fórmula es válida. Presentar el resultado en CNF. (a) La operación XOR (o exclusivo) entre dos variables. (b) La afirmación de que al menos dos de las tres variables A,B,C son verdaderas.

Ejercicio 2.10 Resuelve el siguiente caso con la ayuda de una prueba de resolución: "Si el delincuente tenía un cómplice, entonces venía en un auto. El delincuente no tenía cómplice y no tenía la llave, o tenía la llave y un cómplice. El criminal tenía la llave. ¿El criminal vino en un carro o no?".

Ejercicio 2.11 Muestre por resolución que la fórmula del (a) Ejercicio

2.2(d) es una tautología. (b) El ejercicio

2.4(c) no es satisfactorio.

Ejercicio 2.12 Demuestre las siguientes equivalencias, que son importantes para trabajar con cláusulas de Horn: (a) $(\neg A \vee B) \equiv A \rightarrow B$ (b) $(\neg A \vee \neg B) \equiv A \wedge B$

(c) $(\neg A \vee \neg B) \equiv \neg(A \wedge B)$

(d) $(\neg A \vee \neg B) \equiv \neg(A \wedge B)$

Ejercicio 2.13 Muestre mediante resolución SLD que el siguiente conjunto de cláusulas Horn no se puede cumplir.

(A)1	(D)4	(A $\vee$ D $\vee$ G)7
(B)2	(E)5	(C $\vee$ F $\vee$ E $\vee$ H)8
(C)3	(A $\wedge$ B $\wedge$ C $\wedge$ F)6	(H $\wedge$ f)9

Ejercicio 2.14 En la Secc. 2.6 dice: "Por lo tanto, está claro que probablemente (módulo del problema P/NP) no haya un algoritmo polinomial para 3-SAT y, por lo tanto, probablemente tampoco uno general". Justifique el "probablemente" en esta oración.

Muchos problemas prácticos y relevantes no pueden formularse o sólo pueden formularse de manera muy inconveniente en el lenguaje de la lógica proposicional, como podemos reconocer fácilmente en el siguiente ejemplo. La declaración

"El robot 7 está situado en la posición xy (35, 79)"

de hecho, puede usarse directamente como la variable lógica proposicional

"Robot_7_está_situado_en_posición_xy_(35, 79)"

para razonar con lógica proposicional, pero razonar con este tipo de proposiciones es muy inconveniente. Suponga que 100 de estos robots pueden detenerse en cualquier lugar en una cuadrícula de 100×100 puntos. Para describir cada posición de cada robot, necesitaríamos $100 \cdot 100 \cdot 100 = 1\,000\,000 = 10^6$ variables diferentes. La definición de relaciones entre objetos (aquí robots) se vuelve verdaderamente difícil. La relación

"El robot A está a la derecha del robot B".

semánticamente no es más que un conjunto de pares. De los 10 000 pares posibles de coordenadas x hay $(99 \cdot 98)/2 = 4851$ pares ordenados. Junto con las 10 000 combinaciones de posibles valores de y para ambos robots, hay $(100 \cdot 99) = 9900$ fórmulas del tipo

Robot_7_está_a_la_derecha_de_robot_12

Robot_7_está_situado_en_posición_xy_(35, 79)

Robot_12_está_situado_en_posición_xy_(10, 93) ...

definiendo estas relaciones, cada una de ellas con $(104)^2 \cdot 0,485 = 0,485 \cdot 108$ alternativas en el lado derecho. En la lógica de predicados de primer orden, podemos definir para esto un predicado Position(number, xPosition, yPosition). La relación anterior ya no debe enumerarse como un gran número de pares, sino que se describe de forma abstracta con una regla de la forma

$\forall u \forall v \text{ está\_más\_derecha}(u, v)$

$\exists x_u \exists y_u \exists x_v \exists y_v \text{ posición}(u, x_u, y_u) \text{ posición}(v, x_v, y_v) \quad x_u > x_v,$

Donde $\forall$ se lee como "para todo u" y $\exists$ como "existe v".

En este capítulo definiremos la sintaxis y la semántica de la lógica de predicados de primer orden (PL1), mostraremos que muchas aplicaciones se pueden modelar con este lenguaje y mostraremos que existe un cálculo sólido y completo para este lenguaje.

3.1 Sintaxis

Primero solidificamos la estructura sintáctica de los términos.

Definición 3.1 Sea V un conjunto de variables, K un conjunto de constantes y F un conjunto de símbolos de funciones. Los conjuntos V, K y F son disjuntos por pares. Definimos el conjunto de términos recursivamente:

- Todas las variables y constantes son términos (atómicos).
- Si $t_1, \dots, t_n$ son términos y f un símbolo de función de n lugares, entonces $f(t_1, \dots, t_n)$ también es un término.

Algunos ejemplos de términos son $f(\sin(\ln(3)))$, $\exp(x)$ y $g(g(g(x)))$. Ser capaz de establecemos relaciones lógicas entre términos, construimos fórmulas a partir de términos.

Definición 3.2 Sea P un conjunto de símbolos predicados. Las fórmulas de lógica de predicados se construyen de la

siguiente manera:

- Si $t_1, \dots, t_n$ son términos y p un símbolo de predicado de n posiciones, entonces $p(t_1, \dots, t_n)$ es una fórmula (atómica).
- Si A y B son fórmulas, entonces $\neg A$, $(A) \quad B$, $A \quad B$, $A \quad B$, $A \quad B$ también son fórmulas.
- Si x es una variable y A una fórmula, entonces $\forall x A$ y $\exists x A$ también son fórmulas. $\forall$ es el cuantificador universal y $\exists$ el cuantificador existencial.
- $p(t_1, \dots, t_n)$ y $\neg p(t_1, \dots, t_n)$ se denominan literales.
- Las fórmulas en las que cada variable está en el ámbito de un cuantificador se denominan oraciones de primer orden o fórmulas cerradas. Las variables que no están dentro del alcance de un cuantificador se denominan variables libres.
- Las definiciones 2.8 (CNF) y 2.10 (cláusulas de Horn) son válidas para fórmulas de literales lógicos predicados de manera análoga.

En la Tabla 3.1 en la página 33 se dan varios ejemplos de fórmulas PL1 junto con con sus interpretaciones intuitivas.

Tabla 3.1 Ejemplos de fórmulas en lógica de predicados de primer orden. Tenga en cuenta que la madre aquí es una símbolo de función

Fórmula	Descripción
$\forall x \text{ rana}(x) \rightarrow \text{verde}(x)$	Todas las ranas son verdes.
$\forall x \text{ rana}(x) \rightarrow (\text{marrón}(x) \rightarrow \text{grande}(x))$	Todas las ranas marrones son grandes.
$\forall x \text{ gusta}(x, \text{ pastel})$	a todos les gusta el pastel
$\neg \forall x \text{ me gusta}(x, \text{ pastel})$	No a todos les gusta el pastel.
$\neg \exists x \text{ me gusta}(x, \text{ pastel})$	a nadie le gusta el pastel
$\exists x \forall y \text{ me gusta}(y, x)$	Hay algo que le gusta a todo el mundo.
$\exists x \forall y \text{ me gusta}(x, y)$	Hay alguien a quien le gusta todo.
$\forall y \exists x \text{ me gusta}(y, x)$	Todo es amado por alguien.
$\forall x \exists y \text{ me gusta}(x, y)$	a todos les gusta algo
$\forall y \exists x \text{ cliente}(x) \rightarrow \text{me gusta}(\text{bob}, x)$	A Bob le gustan todos los clientes.
$\exists \text{ cliente}(x) \rightarrow \text{me gusta}(x, \text{ bob})$	Hay un cliente que le gusta a Bob.
$\exists \text{ panadero}(x) \rightarrow \forall y \text{ cliente}(y) \rightarrow \text{mag}(x, y)$	Hay un panadero al que le gustan todos sus clientes
$\forall x \text{ mayor}(\text{madre}(x), x)$	Toda madre es mayor que su hijo.
$\text{mayor}(\text{madre}(\text{madre}(x)), x)$	Toda abuela es mayor que ella. hijo de la hija
$\forall x \forall y \forall z (\text{rel}(x, y) \wedge \text{rel}(y, z) \rightarrow \text{rel}(x, z))$	rel es una relación transitiva

3.2 Semántica

En la lógica proposicional, a cada variable se le asigna directamente un valor de verdad mediante una interpretación. En la lógica de predicados, el significado de las fórmulas se define recursivamente sobre la construcción de la fórmula, en la que primero asignamos constantes, variables y función símbolos a objetos en el mundo real.

Definición 3.3 Una interpretación I se define como

- Un mapeo del conjunto de constantes y variables $K \cup V$ a un conjunto W de nombres de objetos en el mundo.
- Un mapeo del conjunto de símbolos de función al conjunto de funciones en el mundo. A cada símbolo de función de n posiciones se le asigna una función de n posiciones.
- Un mapeo del conjunto de símbolos de predicado al conjunto de relaciones en el mundo. A cada símbolo de predicado de n lugares se le asigna una relación de n lugares.

Ejemplo 3.1 Sean constantes c_1, c_2, c_3 , “más” un símbolo de función de dos posiciones y “gr” un símbolo de predicado de dos lugares. La verdad de la fórmula.

$$F \equiv \text{gr}(\text{más}(c_1, c_3), c_2)$$

depende de la interpretación I. Primero elegimos la siguiente interpretación obvia de las constantes, la función y los predicados en los números naturales:

$$I1: c1 \rightarrow 1, c2 \rightarrow 2, c3 \rightarrow 3, \text{ más} \rightarrow +, \text{ gr} \rightarrow >$$

Por lo tanto, la fórmula se asigna a

$$1 + 3 > 2, \text{ o después de la evaluación } 4 > 2.$$

La relación mayor que en el conjunto $\{1, 2, 3, 4\}$ es el conjunto de todos los pares (x, y) de números con $x > y$, lo que significa que el conjunto $G = \{(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)\}$. Como $(4, 2) \in G$, la fórmula F es verdadera bajo la interpretación I1. Sin embargo, si optamos por la interpretación

$$I2: c1 \rightarrow 2, c2 \rightarrow 3, c3 \rightarrow 1, \text{ más} \rightarrow -, \text{ gr} \rightarrow >,$$

obtenemos

$$2 - 1 > 3, \text{ o } 1 > 3.$$

El par $(1, 3)$ no es miembro de G. La fórmula F es falsa bajo la interpretación I2. Obviamente, la verdad de una fórmula en PL1 depende de la interpretación. Ahora, después de esta vista previa, definimos la verdad.

Definición 3.4 •

Una fórmula atómica $p(t_1, \dots, t_n)$ es verdadera (o válida) bajo la interpretación I si, después de la interpretación y evaluación de todos los términos $t_1, \dots, t_n$ y la interpretación del predicado p hasta la relación r de n lugares, se cumple que

$$(I(t_1), \dots, I(t_n)) \quad r.$$

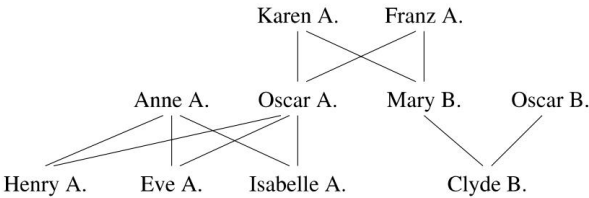
- La verdad de las fórmulas sin cuantificadores se deriva de la verdad de las fórmulas atómicas, como en el cálculo proposicional, a través de la semántica de los operadores lógicos definidos en la Tabla 2.1 en la página 17. •

Una fórmula $\exists x F$ es verdadera bajo la interpretación I exactamente cuando es verdadera dado un cambio arbitrario de la interpretación de la variable x (y solo para x)

- Una fórmula $\exists x F$ es verdadera bajo la interpretación I exactamente cuando hay una interpretación para x que hace que la fórmula sea verdadera.

Las definiciones de equivalencia semántica de fórmulas, para los conceptos satisficible, verdadero, insatisficible y modelo, junto con la implicación semántica (Definiciones 2.4, 2.5, 2.6) se trasladan sin cambios del cálculo proposicional a la lógica de predicados.

Fig. 3.1 Un árbol genealógico.
Los bordes que van desde
Clyde B. hacia arriba hasta
Mary B. y Oscar B.
representan el elemento (Clyde
B., Mary B., Oscar B.)
como una relación secundaria



Teorema 3.1 Los teoremas 2.2 (teorema de deducción) y 2.3 (demostración por contradicción) se cumplen análogamente para PL1.

Ejemplo 3.2 El árbol genealógico de la figura 3.1 representa gráficamente (en el nivel semántico) la relación

Niño = {(Oscar A., Karen A., Frank A.), (Mary B., Karen A., Frank A.), (Henry A., Anne A., Oscar A.), (Eve A., Anne A., Oscar A.), (Isabelle A., Anne A., Oscar A.), (Clyde B., Mary B., Oscar B.)}

Por ejemplo, el triple (Oscar A., Karen A., Frank A.) representa la proposición “Oscar A. es hijo de Karen A. y Frank A.”. De los nombres leemos la relación de un lugar

Mujer = {Karen A., Anne A., Mary B., Eve A., Isabelle A.}

de las mujeres Ahora queremos establecer fórmulas para las relaciones familiares. Primero definimos un niño predicado de tres lugares (x, y, z) con la semántica

$I(\text{niño}(x, y, z)) = w \equiv (I(x), I(y), I(z)) \quad \text{Tipo.}$

Bajo la interpretación $I(\text{oscar}) = \text{Oscar A.}$, $I(\text{eve}) = \text{Eve A.}$, $I(\text{anne}) = \text{Anne A.}$, también es cierto que $\text{child}(\text{eve}, \text{anne}, \text{oscar})$. Para que $\text{child}(\text{eve}, \text{oscar}, \text{anne})$ sea verdad, requerimos, con

$x \quad y \quad z \quad \text{hijo}(x, y, z) \quad \text{hijo}(x, z, y),$

simetría del predicado hijo en los dos últimos argumentos. Para más definiciones nos referimos al Ejercicio 3.1 en la página 54 y definimos el predicado descendiente recursivamente

como

$x \quad y \quad \text{descendiente}(x, y) \quad z \quad \text{hijo}(x, y, z)$
 $(\quad u \quad v \quad \text{hijo}(x, u, v) \quad \text{descendiente}(u, y)).$

Ahora construimos una pequeña base de conocimiento con reglas y hechos. Dejar

$KB \equiv \text{mujer}(\text{karen}) \quad \text{mujer}(\text{ana}) \quad \text{mujer}(\text{maria})$
 $\text{mujer}(\text{eva}) \quad \text{mujer}(\text{isabelle})$
 $\text{hijo}(\text{oscar}, \text{karen}, \text{franz}) \quad \text{hijo}(\text{maria}, \text{karen}, \text{franz})$
 $\text{hijo}(\text{eva}, \text{ana}, \text{oscar}) \quad \text{hijo}(\text{enrique}, \text{ana}, \text{oscar})$
 $\text{niño}(\text{isabelle}, \text{anne}, \text{oscar}) \quad \text{niño}(\text{clyde}, \text{mary}, \text{oscar})$
 $(\ x \ y \ z \text{ hijo}(x, y, z) \quad \text{hijo}(x, z, y))$
 $(\ x \ y \text{ descendiente}(x, y) \quad z \text{ hijo}(x, y, z)$
 $(\ u \ v \text{ hijo}(x, u, v) \quad \text{descendiente}(u, y)))$.

Ahora podemos preguntar, por ejemplo, si las proposiciones $\text{niño}(\text{eva}, \text{oscar}, \text{ana})$ o $\text{descendiente}(\text{eva}, \text{franz})$ son derivables. Para ello necesitamos un cálculo.

3.2.1 Igualdad

Para poder comparar términos, la igualdad es una relación muy importante en la lógica de predicados. La igualdad de términos en matemáticas es una relación de equivalencia, lo que significa que es reflexiva, simétrica y transitiva. Si queremos usar la igualdad en fórmulas, debemos incorporar estos tres atributos como axiomas en nuestra base de conocimientos o debemos integrar la igualdad en el cálculo. Tomamos el camino fácil y definimos un predicado "=" que, a diferencia de la Definición 3.2 en la página 32, se escribe usando notación infija como es costumbre en matemáticas. (Por supuesto, una ecuación $x = y$ también podría escribirse en la forma $\text{eq}(x, y)$.) Por lo tanto, los axiomas de igualdad tienen la forma

$$\begin{array}{ll}
 x = x & \text{(reflexividad)} \\
 y = x \quad x = y & \text{(simetría)} \\
 y = z \quad x = y \quad x = z & \text{(transitividad)}.
 \end{array} \tag{3.1}$$

Para garantizar la unicidad de las funciones, requerimos adicionalmente

$$x = y \quad f(x) = f(y) \quad \text{(axioma de sustitución)} \tag{3.2}$$

para cada símbolo de función. Análogamente requerimos para todos los símbolos de predicado

$$x = y \quad p(x) \quad p(y) \quad \text{(axioma de sustitución)}. \tag{3.3}$$

Formulamos otras relaciones matemáticas, como la relación "<", por medios similares (Ejercicio 3.4 en la página 55).

A menudo, una variable debe ser reemplazada por un término. Para llevarlo a cabo correctamente y describirlo simplemente, damos la siguiente definición.

fórmula sin cuantificadores. • $Q_i \{ \text{ , } \}$
 $\}$ para $i = 1, \dots, n$.

Undergraduate Topics in Computer Science

Wolfgang Ertel

Introduction to Artificial Intelligence



 Springer

Temas de Pregrado en Ciencias de la Computación

Temas de pregrado en ciencias de la computación (UTiCS) ofrece contenido instructivo de alta calidad para estudiantes de pregrado que estudian en todas las áreas de informática y ciencias de la información. Desde material fundamental y teórico básico hasta temas y aplicaciones del último año, los libros de UTiCS adoptan un enfoque fresco, conciso y moderno y son ideales para el autoaprendizaje o para un curso de uno o dos semestres. Todos los textos están escritos por expertos establecidos en sus campos, revisados por un consejo asesor internacional y contienen numerosos ejemplos y problemas. Muchos incluyen soluciones completamente trabajadas.

Para más volúmenes:

www.springer.com/series/7592

Wolfgang Ertel

Introducción a los artificiales Inteligencia

Traducido por Natanael Black
Con ilustraciones de Florian Mast

Prof. Dr. Wolfgang Ertel FB
Elektrotechnik und Informatik Hochschule
Ravensburg-Weingarten Universidad de
Ciencias Aplicadas Weingarten
Alemania
[ertel@hs-
weingarten.de](mailto:ertel@hs-weingarten.de)

Editor de series
Ian Mackie

junta asesora

Samson Abramsky, Universidad de Oxford, Oxford, Reino Unido
Karin Breitman, Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Brasil
Chris Hankin, Imperial College London, Londres, Reino Unido
Dexter Kozen, Universidad de Cornell, Ithaca, EE. UU.
Andrew Pitts, Universidad de Cambridge, Cambridge, Reino Unido
Hanne Riis Nielson, Universidad Técnica de Dinamarca, Kongens Lyngby, Dinamarca
Steven Skiena, Universidad de Stony Brook, Stony Brook, EE. UU.
Iain Stewart, Universidad de Durham, Durham, Reino Unido

ISSN 1863-7310

ISBN 978-0-85729-298-8 DOI

e-ISBN 978-0-85729-299-5

10.1007/978-0-85729-299-5 Springer

Londres Dordrecht Heidelberg Nueva York

Catalogación de la Biblioteca Británica en datos de publicación
Un registro de catálogo para este libro está disponible en la Biblioteca Británica.

Número de control de la Biblioteca del Congreso: 2011923216

Publicado originalmente en alemán por Vieweg+Teubner, 65189 Wiesbaden, Alemania, como "Ertel, Wolfgang: Grundkurs Künstliche Intelligenz. 2. rev. edición". © Vieweg+Teubner | GWV Fachverlage GmbH, Wiesbaden 2009

© Springer-Verlag London Limited 2011 Aparte de

cualquier trato justo con fines de investigación o estudio privado, o crítica o revisión, según lo permitido por la Ley de derechos de autor, diseños y patentes de 1988, esta publicación solo puede reproducirse, almacenarse o transmitirse, en cualquier forma o por cualquier medio, con el permiso previo por escrito de los editores, o en el caso de la reproducción reprográfica de acuerdo con los términos de las licencias emitidas por la Agencia de Licencias de Derechos de Autor. Las consultas relativas a la reproducción fuera de esos términos deben enviarse a los editores.

El uso de nombres registrados, marcas comerciales, etc., en esta publicación no implica, incluso en ausencia de una declaración específica, que dichos nombres estén exentos de las leyes y reglamentos pertinentes y, por lo tanto, libres para uso general.

El editor no hace ninguna representación, expresa o implícita, con respecto a la exactitud de la información contenida en este libro y no puede aceptar ninguna responsabilidad legal por cualquier error u omisión que pueda cometerse.

Impreso en papel libre de ácido

Springer es parte de Springer Science+Business Media (www.springer.com)

Prefacio

La Inteligencia Artificial (IA) tiene el objetivo definido de comprender la inteligencia y construir sistemas inteligentes. Sin embargo, los métodos y formalismos utilizados en el camino hacia este objetivo no están firmemente establecidos, lo que ha dado como resultado que la IA consista en una multitud de subdisciplinas en la actualidad. La dificultad en un curso introductorio de IA radica en transmitir tantas ramas como sea posible sin perder demasiada profundidad y precisión.

El libro de Russell y Norvig [RN10] es más o menos la introducción estándar a la IA. Sin embargo, dado que este libro tiene 1152 páginas, y dado que es demasiado extenso y costoso para la mayoría de los estudiantes, los requisitos para escribir este libro eran claros: debería ser una introducción accesible a la IA moderna para el autoaprendizaje o como la base de un curso de una hora, con un máximo de 300 páginas. El resultado está frente a ti.

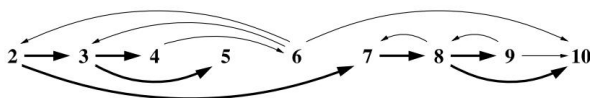
En el espacio de 300 páginas, un campo tan extenso como la IA no puede cubrirse por completo. Para evitar convertir el libro en un índice, he intentado profundizar un poco e introducir algoritmos y aplicaciones concretas en cada una de las siguientes ramas: agentes, lógica, búsqueda, razonamiento con incertidumbre, aprendizaje automático y redes neuronales.

Los campos de procesamiento de imágenes, lógica difusa y procesamiento de lenguaje natural no se tratan en detalle. El campo del procesamiento de imágenes, que es importante para todas las ciencias de la computación, es una disciplina independiente con muy buenos libros de texto, como [GW08]. El procesamiento del lenguaje natural tiene un estado similar. Al reconocer y generar texto y lenguaje hablado, se aplican métodos de lógica, razonamiento probabilístico y redes neuronales. En este sentido este campo es parte de la IA. Por otro lado, la lingüística informática es una rama extensa de la informática y tiene mucho en común con los lenguajes formales. En este libro señalaremos tales sistemas apropiados en varios lugares, pero no daremos una introducción sistemática. Para una primera introducción en este campo, nos remitimos a los Caps. 22 y 23 en [RN10]. La lógica difusa, o teoría de conjuntos difusos, se ha convertido en una rama de la teoría de control debido a su aplicación principal en la tecnología de automatización y se trata en los libros y conferencias correspondientes.

Por lo tanto, renunciaremos a una introducción aquí.

Las dependencias entre los capítulos del libro están esbozadas a grandes rasgos en el gráfico que se muestra a continuación. Para mantenerlo simple, Cap. 1, con la introducción fundamental para todos los capítulos siguientes, se omite. Como ejemplo, la flecha más gruesa de 2 a 3 significa que la lógica proposicional es un requisito previo para comprender la lógica de predicados. La flecha delgada del 9 al 10 significa que las redes neuronales son útiles para comprender el aprendizaje por refuerzo, pero no son absolutamente necesarias. Delgado hacia atrás

las flechas deben dejar en claro que los capítulos posteriores pueden brindar una comprensión más profunda a temas que ya han sido aprendidos.



Este libro es aplicable a estudiantes de informática y otras ciencias naturales técnicas. ciencias y, en su mayor parte, requiere conocimientos de matemáticas de nivel secundario. En varios lugares, el conocimiento del álgebra lineal y el análisis multidimensional se necesita Para una comprensión más profunda de los contenidos, es indispensable trabajar activamente los ejercicios. Esto significa que las soluciones sólo deben consultarse después de un trabajo intensivo con cada problema, y sólo para comprobar las propias soluciones, fiel a Lema de Leonardo da Vinci "Estudiar sin devoción daña el cerebro". Un poco los problemas más difíciles se marcan con y los especialmente difíciles con . Los problemas que requieren conocimientos especiales de programación o informática son marcado con .

En el sitio web del libro en www.hs-weingarten.de/~ertel/aibook materiales digitales para los ejercicios, como datos de entrenamiento para aprender algoritmos, una página con referencias a los programas de IA mencionados en el libro, una lista de enlaces a los temas tratados, un lista de la bibliografía en la que se puede hacer clic, una lista de erratas y diapositivas de presentación para profesores puede ser encontrado. Ruego al lector enviar sugerencias, críticas y tips sobre errores directamente a ertel@hs-weingarten.de.

Este libro es una traducción actualizada de mi libro alemán "Grundkurs Künstliche Intelligenz" publicado por Vieweg Verlag. Mi agradecimiento especial al traductor. Nathan Black, quien en una excelente cooperación transatlántica entre Alemania y California a través de SVN, Skype y correo electrónico produjo este texto. Estoy agradecido con Franz Kur feß, quien me presentó a Nathan; a Matthew Wight por revisar la traducción libro y a Simon Rees de Springer Verlag por su paciencia.

Quisiera agradecer a mi esposa Evelyn por su apoyo y paciencia durante este proyecto que consume mucho tiempo. Un agradecimiento especial a Wolfgang Bibel y Chris Lobenschuss, quienes corrigieron cuidadosamente el manuscrito alemán. Sus sugerencias y discusiones conducen a muchas mejoras y adiciones. Para leer las correcciones y otros valiosos servicios, me gustaría agradecer a Richard Cubek, Celal Döven, Joachim Feßler, Nico Hochgeschwender, Paul Kirner, Wilfried Meister, Norbert Perk, Peter Radtke, Markus Schneider, Manfred Schramm, Uli Stärk, Michel Tokic, Arne Usadel y todos los estudiantes interesados. También doy las gracias a Florian Mast por la viñetas de valor incalculable y una colaboración muy eficaz.

Espero que durante sus estudios este libro le ayude a compartir mi fascinación por Inteligencia artificial.

Contenido

1	Introducción	1
1.1	¿Qué es la Inteligencia Artificial?	1
1.1.1	Ciencias del cerebro y resolución de problemas	3
1.1.2	La prueba de Turing y los chatterbots	4
1.2	La historia de la IA	5
1.2.1	Los primeros comienzos	5
1.2.2	La lógica resuelve (casi) todos los problemas	6
1.2.3	El nuevo conexionismo	7
1.2.4	Razonamiento bajo incertidumbre	7
1.2.5	Agentes distribuidos, autónomos y de aprendizaje	8
1.2.6	La IA crece	9
1.3	Agentes	9
1.4	Sistemas basados en el conocimiento	12
1.5	Ejercicios	14
2	Lógica proposicional	15
2.1	Sintaxis	15
2.2	Semántica	dieciséis
2.3	Sistemas de prueba	18
2.4	Resolución	22
2.5	Cláusulas Horn	25
2.6	Computabilidad y Complejidad	27
2.7	Aplicaciones y limitaciones	28
2.8	Ejercicios	29
3	Lógica de predicados de primer orden	31
3.1	Sintaxis	32
3.2	Semántica	33
3.2.1	Igualdad	36
3.3	Cuantificadores y formas normales	37
3.4	Cálculos de prueba	40
3.5	Resolución	42
3.5.1	Estrategias de resolución	46
3.5.2	Igualdad	46
3.6	Demostradores automáticos de teoremas	47

3.7 Ejemplos matemáticos. . . 3.8	48
Aplicaciones.	51
3.9 Resumen.	54
3.10 Ejercicios	54
4 Limitaciones de la Lógica.	57
4.1 El problema del espacio de	57
búsqueda 4.2 Decidibilidad e incompletitud	59
4.3 El pingüino volador.	60
4.4 Modelado de la incertidumbre.	63
4.5 Ejercicios	sesenta y cinco
5 Programación Lógica con PROLOG	67
5.1 Sistemas e implementaciones PROLOG.	68
5.2 Ejemplos simples.	68
5.3 Control de ejecución y elementos de procedimiento	71
5.4 Listas	73
5.5 Programas automodificables.	75
5.6 Un ejemplo de planificación.	76
Programación Lógica de Restricciones.	77
5.7 5.8 Resumen	79
5.9 Ejercicios	80
6 Búsqueda, juegos y resolución de problemas	83
6.1 Introducción 6.2	83
Búsqueda no informada.	89
6.2.1 Búsqueda en amplitud.	89
6.2.2 Búsqueda en profundidad primero	91
6.2.3 Profundización iterativa.	92
6.2.4 Comparación.	94
6.3 Búsqueda heurística.	94
6.3.1 Búsqueda codiciosa.	96
6.3.2 A -Buscar	98
6.3.3 IDA -Buscar	100
6.3.4 Comparación empírica de los algoritmos de búsqueda.	100
6.3.5 Resumen.	102
6.4 Juegos con Oponentes.	102
6.4.1 Búsqueda Minimax.	103
6.4.2 Poda alfa-beta.	103
6.4.3 Juegos no deterministas.	106
6.5 Funciones de Evaluación Heurística	106
6.5.1 Aprendizaje de Heurísticas	107
6.6 Ejercicios de	108
6.7 Estado del Arte	110
7 Razonamiento con incertidumbre	113
7.1 Computación con probabilidades	115

7.1.1 Probabilidad Condicional.	118
7.2 El Principio de Máxima Entropía.	122
7.2.1 Una regla de inferencia para	123
probabilidades 7.2.2 Entropía máxima sin restricciones explícitas.	127
7.2.3 Probabilidad Condicional Versus Implicación Material.	128
7.2.4 Sistemas MaxEnt	129
7.2.5 El ejemplo de Piolín.	130
7.3 LEXMED, un Sistema Experto para el Diagnóstico de Apendicitis.	131
7.3.1 Diagnóstico de Apendicitis con Métodos Formales	131
7.3.2 Base de Conocimiento Probabilística Híbrida	132
7.3.3 Aplicación de LEXMED 7.3.4	135
Función de LEXMED 7.3.5	136
Gestión de Riesgos Utilizando la Matriz de Costos	139
7.3.6 Desempeño	141
7.3.7 Áreas de Aplicación y Experiencias.	143
7.4 Razonamiento con Redes Bayesianas.	144
7.4.1 Variables independientes.	144
7.4.2 Representación gráfica del conocimiento como bayesiano Red	146
7.4.3 Independencia Condicional.	146
7.4.4 Aplicación práctica.	148
7.4.5 Software para Redes Bayesianas.	149
7.4.6 Desarrollo de Redes Bayesianas.	150
7.4.7 Semántica de Redes Bayesianas.	154
7.5 Resumen. 7.6	156
Ejercicios	157
8 Aprendizaje automático y minería de datos.	161
8.1 Análisis de datos.	166
8.2 El perceptrón, un clasificador lineal.	169
8.2.1 La regla de	171
aprendizaje 8.2.2 Optimización y perspectiva.	174
8.3 El método del vecino más cercano	175
8.3.1 Dos clases, muchas clases, aproximación.	179
8.3.2 La distancia es relevante.	180
8.3.3 Tiempos de Cómputo	181
8.3.4 Resumen y Perspectiva.	182
8.3.5 Razonamiento basado en casos.	183
8.4 Aprendizaje del árbol de decisiones.	184
8.4.1 Un ejemplo sencillo.	185
8.4.2 La entropía como métrica del contenido de la información.	186
8.4.3 Ganancia de información.	189
8.4.4 Aplicación de C4.5.	191
8.4.5 Aprendizaje del Diagnóstico de Apendicitis.	193
8.4.6 Atributos continuos	196

8.4.7 Poda—Cortar el árbol 8.4.8	197
Valores faltantes	198
8.4.9 Resumen.	199
8.5 Aprendizaje de Redes Bayesianas.	199
8.5.1 Aprendizaje de la estructura de la red.	199
8.6 El clasificador Naive Bayes.	202
8.6.1 Clasificación de texto con Naive Bayes.	204
8.7 Agrupación.	206
8.7.1 Métricas de	207
distancia 8.7.2 k-Means y el algoritmo EM.	208
8.7.3 Agrupación jerárquica.	209
8.8 Minería de datos en la práctica	211
8.8.1 La herramienta de minería de datos KNIME.	212
8.9 Resumen.	214
8.10 Ejercicios	216
8.10.1 Introducción	216
8.10.2 El Perceptrón	216
8.10.3 Método del vecino más	217
cercano 8.10.4 Árboles	218
de decisión 8.10.5 Aprendizaje de redes bayesianas.	219
8.10.6 Agrupación.	220
8.10.7 Minería de datos.	220
9 Redes Neuronales	221
9.1 De la biología a la simulación.	222
9.1.1 El modelo matemático 9.2	223
Redes de Hopfield.	226
9.2.1 Aplicación a un ejemplo de reconocimiento de patrones.	227
9.2.2 Análisis.	228
9.2.3 Resumen y perspectiva.	231
9.3 Memoria Asociativa Neural.	232
9.3.1 Memoria de matriz de correlación.	233
9.3.2 La pseudoinversa.	235
9.3.3 La regla hebb binaria	236
9.3.4 Un programa de corrección ortográfica	238
9.4 Redes lineales con errores mínimos 9.4.1	240
Método de mínimos cuadrados	241
9.4.2 Aplicación a los datos de apendicitis	242
9.4.3 La regla delta	243
9.4.4 Comparación con el perceptrón.	245
9.5 El algoritmo de retropropagación.	246
9.5.1 NETtalk: una red aprende a hablar	249
9.5.2 Aprendizaje de heurísticas para probadores de teoremas.	250
9.5.3 Problemas y mejoras.	251
9.6 Máquinas de vectores de soporte	252

9.7 Aplicaciones.	254
9.8 Resumen y perspectiva. . 9.9	254
Ejercicios 9.9.1	255
De la biología a la simulación.	255
9.9.2 Redes Hopfield.	255
9.9.3 Redes lineales con errores mínimos	255
9.9.4 Retropropagación.	256
9.9.5 Máquinas de vectores de soporte	256
10 Aprendizaje por refuerzo	257
10.1 Introducción 10.2	257
La tarea 10.3	259
Búsqueda combinatoria no informada	261
10.4 Iteración de valor y programación dinámica.	262
10.5 Un robot que camina aprendiendo y su simulación	265
10.6 Q-Learning.	267
10.6.1 Q-Learning en un entorno no determinista.	270
10.7 Exploración y Explotación.	272
10.8 Aproximación, generalización y convergencia.	272
10.9 Aplicaciones.	273
10.10 Maldición de dimensionalidad.	274
10.11 Resumen y perspectiva.	275
10.12 Ejercicios	276
11 Soluciones de los Ejercicios	279
11.1 Introducción	279
11.2 Lógica proposicional.	280
11.3 Lógica de predicados de primer orden.	282
11.4 Limitaciones de la lógica.	283
11.5 PRÓLOGO.	283
11.6 Búsqueda, juegos y resolución de problemas.	285
11.7 Razonamiento con incertidumbre.	287
11.8 Aprendizaje automático y minería de datos.	293
11.9 Redes neuronales.	300
11.10 Aprendizaje por refuerzo.	301
Referencias	305
índice	311

Introducción

1

1.1 ¿Qué es la Inteligencia Artificial?

El término inteligencia artificial despierta emociones. Por un lado, está nuestra fascinación por la inteligencia, que aparentemente nos otorga a los humanos un lugar especial entre las formas de vida. Surgen preguntas como "¿Qué es la inteligencia?", "¿Cómo se puede medir la inteligencia?" o "¿Cómo funciona el cerebro?". Todas estas preguntas son significativas cuando se trata de entender la inteligencia artificial. Sin embargo, la cuestión central para el ingeniero, especialmente para el informático, es la cuestión de la máquina inteligente que se comporta como una persona, mostrando un comportamiento inteligente.

El atributo artificial puede despertar asociaciones muy diferentes. Trae miedo a los cyborgs inteligentes. Recuerda imágenes de novelas de ciencia ficción. Plantea la cuestión de si nuestro mayor bien, el alma, es algo que deberíamos tratar de entender, modelar o incluso reconstruir.

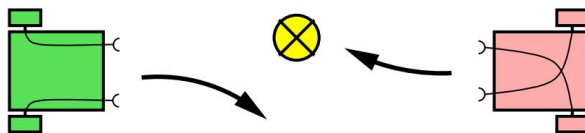
Con interpretaciones tan diferentes e improvisadas, se vuelve difícil definir el término inteligencia artificial o IA de manera simple y sólida. Sin embargo, me gustaría intentar, utilizando ejemplos y definiciones históricas, caracterizar el campo de la IA. En 1955, John McCarthy, uno de los pioneros de la IA, fue el primero en definir el término inteligencia artificial, más o menos de la siguiente manera:

El objetivo de la IA es desarrollar máquinas que se comporten como si fueran inteligentes.

Para probar esta definición, el lector puede imaginar el siguiente escenario. Una quincena de pequeños vehículos robóticos se mueven en una superficie cuadrada cerrada de cuatro por cuatro metros. Uno puede observar varios patrones de comportamiento. Algunos vehículos forman pequeños grupos con relativamente poco movimiento. Otros se mueven tranquilamente por el espacio y evitan con gracia cualquier colisión. Aún otros parecen seguir a un líder. Los comportamientos agresivos también son observables. ¿Es lo que estamos viendo un comportamiento inteligente?

Según la definición de McCarthy, los robots antes mencionados pueden describirse como inteligentes. El psicólogo Valentin Braitenberg ha demostrado que este comportamiento aparentemente complejo puede ser producido por circuitos eléctricos muy simples [Bra84]. Los llamados vehículos Braitenberg tienen dos ruedas, cada una de las cuales es impulsada por un motor eléctrico independiente. La velocidad de cada motor está influenciada por un sensor de luz en

Fig. 1.1 Dos vehículos de Braitenberg muy simples y sus reacciones a una fuente de luz



la parte delantera del vehículo como se muestra en la Fig. 1.1. Cuanta más luz llega al sensor, más rápido funciona el motor. El vehículo 1 de la parte izquierda de la figura, según su configuración, se aleja de una fuente de luz puntual. El vehículo 2, por otro lado, se mueve hacia la fuente de luz. Pequeñas modificaciones adicionales pueden crear otros patrones de comportamiento, de modo que con estos vehículos tan simples podemos realizar el impresionante comportamiento descrito anteriormente.

Claramente, la definición anterior es insuficiente porque la IA tiene como objetivo resolver problemas prácticos difíciles que seguramente son demasiado exigentes para el vehículo de Braitenberg. En la Enciclopedia Britannica [Bri91] se encuentra una Definición que dice así:

La IA es la capacidad de las computadoras digitales o los robots controlados por computadora para resolver problemas que normalmente se asocian con las capacidades de procesamiento intelectual superiores de los humanos.

Pero esta definición también tiene debilidades. Admitiría, por ejemplo, que una computadora con una gran memoria que puede guardar un texto largo y recuperarlo a pedido muestra capacidades inteligentes, ya que la memorización de textos largos ciertamente puede considerarse una capacidad de procesamiento intelectual más alta de los humanos, al igual que, por ejemplo, la capacidad de procesamiento rápido . multiplicación de dos números de 20 dígitos. De acuerdo con esta definición, entonces, cada computadora es un sistema de IA. Este dilema se resuelve elegantemente con la siguiente definición de Elaine Rich [Ric83]:

La inteligencia artificial es el estudio de cómo hacer que las computadoras hagan cosas en las que, en este momento, las personas son mejores.

Rico, breve y conciso, caracteriza lo que los investigadores de IA han estado haciendo durante los últimos 50 años. Incluso en el año 2050, esta definición estará actualizada.

Tareas como la ejecución de muchos cálculos en un corto período de tiempo son los puntos fuertes de las computadoras digitales. En este sentido, superan a los humanos por muchos múltiplos. En muchas otras áreas, sin embargo, los humanos son muy superiores a las máquinas. Por ejemplo, una persona que entra en una habitación desconocida reconocerá el entorno en fracciones de segundo y, si es necesario, tomará decisiones y planificará acciones con la misma rapidez. Hasta la fecha, esta tarea es demasiado exigente para los robots autónomos¹. Según la definición de Rich, esta es, por lo tanto, una tarea para la IA. De hecho, la investigación sobre robots autónomos es un tema importante y actual en IA. La construcción de computadoras de ajedrez, por otro lado, ha perdido relevancia porque ya juegan al nivel de los grandes maestros o por encima de él.

Sin embargo, sería peligroso concluir de la definición de Rich que la IA solo se preocupa por la implementación pragmática de procesos inteligentes. Los sistemas inteligentes, en el sentido de la definición de Rich, no pueden construirse sin una profunda

¹Un robot autónomo funciona de forma independiente, sin apoyo manual, en particular sin control remoto control.

Comprender el razonamiento humano y la acción inteligente en general, por lo que la neurociencia (ver Secc. 1.1.1) es de gran importancia para la IA. Esto también muestra que las otras definiciones citadas reflejan aspectos importantes de la IA.

Una fortaleza particular de la inteligencia humana es la adaptabilidad. Somos capaces de adaptarnos a diversas condiciones ambientales y cambiar nuestro comportamiento en consecuencia a través del aprendizaje. Precisamente porque nuestra capacidad de aprendizaje es muy superior a la de las computadoras, el aprendizaje automático es, según la definición de Rich, un subcampo central de la IA.

1.1.1 Ciencias del cerebro y resolución de problemas

A través de la investigación de sistemas inteligentes podemos tratar de entender cómo funciona el cerebro humano y luego modelarlo o simularlo en la computadora. Muchas ideas y principios en el campo de las redes neuronales (ver el capítulo 9) surgen de la ciencia del cerebro con el campo relacionado de la neurociencia.

Un enfoque muy diferente resulta de tomar una línea de acción orientada a un objetivo, partiendo de un problema y tratando de encontrar la solución más óptima. La forma en que los seres humanos resuelven el problema no tiene importancia aquí. El método, en este enfoque, es secundario. Lo primero y más importante es la solución inteligente óptima para el problema. En lugar de emplear un método fijo (como, por ejemplo, la lógica de predicados), la IA tiene como objetivo constante la creación de agentes inteligentes para tantas tareas diferentes como sea posible. Debido a que las tareas pueden ser muy diferentes, no sorprende que los métodos empleados actualmente en la IA a menudo también sean bastante diferentes. Al igual que la medicina, que abarca muchos procedimientos de diagnóstico y terapia diferentes, que a menudo salvan vidas, la IA también ofrece una amplia gama de soluciones efectivas para una amplia variedad de aplicaciones. Para inspiración mental, considere la Fig. 1.2 en la página 4. Al igual que en medicina, no existe un método universal para todas las áreas de aplicación de la IA, sino una gran cantidad de posibles soluciones para la gran cantidad de problemas cotidianos, grandes y pequeños.

La ciencia cognitiva se dedica a la investigación del pensamiento humano a un nivel algo superior. De manera similar a la ciencia del cerebro, este campo proporciona a la IA práctica muchas ideas importantes. Por otro lado, los algoritmos y las implementaciones conducen a otras conclusiones importantes sobre cómo funciona el razonamiento humano. Así estos tres campos se benefician de un fructífero intercambio interdisciplinario. El tema de este libro, sin embargo, es principalmente la IA orientada a problemas como una subdisciplina de la informática.

Hay muchas cuestiones filosóficas interesantes en torno a la inteligencia y la inteligencia artificial. Los humanos tenemos conciencia; es decir, podemos pensar en nosotros mismos e incluso ponderar que somos capaces de pensar en nosotros mismos. ¿Cómo surge la conciencia? Muchos filósofos y neurólogos creen ahora que la mente y la conciencia están vinculadas con la materia, es decir, con el cerebro. La cuestión de si las máquinas algún día podrían tener una mente o una conciencia podría volverse relevante en algún momento en el futuro. El problema mente-cuerpo en particular se refiere a si la mente está ligada al cuerpo o no. No discutiremos estas cuestiones aquí. El lector interesado puede consultar [Spe98, Spe97] y está invitado, en el curso de los estudios de tecnología de IA, a formarse una opinión personal sobre estas cuestiones.

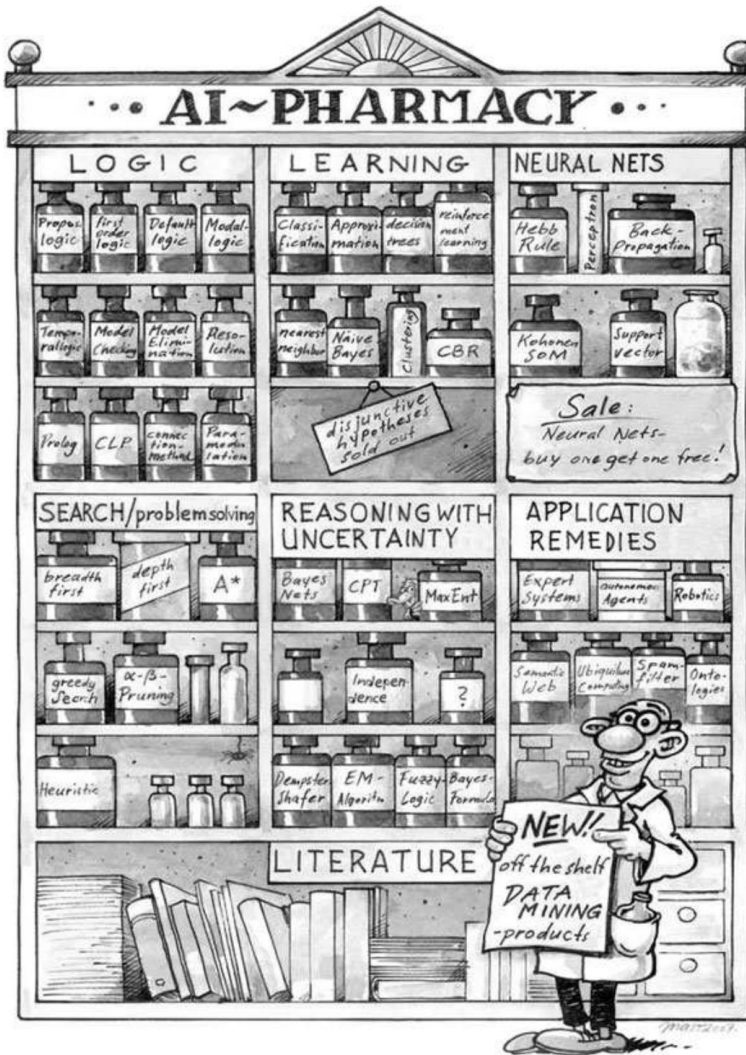


Fig. 1.2 Una pequeña muestra de las soluciones que ofrece la IA

1.1.2 La prueba de Turing y los chatterbots

Alan Turing se hizo un nombre como uno de los primeros pioneros de la IA con su definición de una máquina inteligente, en la que la máquina en cuestión debe pasar la siguiente prueba. La persona de prueba Alice se sienta en una habitación cerrada con dos terminales de computadora. Un terminal está conectado a una máquina, el otro con una persona no maliciosa, Bob. Alice puede escribir preguntas en ambos terminales. Se le da la tarea de decidir, después de cinco minutos, qué terminal pertenece a la máquina. La máquina pasa la prueba si puede engañar a Alice al menos el 30 % de las veces [Tur50].

Si bien la prueba es muy interesante desde el punto de vista filosófico, para la IA práctica, que se ocupa de la resolución de problemas, no es una prueba muy relevante. Las razones de esto son similares a las mencionadas anteriormente en relación con los vehículos de Braitenberg (vea el Ejercicio 1.3 en la página 14).

El pionero de la IA y crítico social Joseph Weizenbaum desarrolló un programa llamado Eliza, que está destinado a responder las preguntas de un sujeto de prueba como un psicólogo humano [Wei66]. De hecho, pudo demostrar el éxito en muchos casos. Supuestamente, su secretaria a menudo tenía largas discusiones con el programa. Hoy en Internet existen muchos de los llamados chatterbots, algunas de cuyas respuestas iniciales son bastante impresionantes. Sin embargo, después de cierto tiempo, su naturaleza artificial se hace evidente. Algunos de estos programas son realmente capaces de aprender, mientras que otros poseen un conocimiento extraordinario de varios temas, por ejemplo, geografía o desarrollo de software. Ya existen aplicaciones comerciales para chatterbots en atención al cliente en línea y puede haber otras en el campo del aprendizaje electrónico. Es concebible que el alumno y el sistema de aprendizaje electrónico puedan comunicarse a través de un chatterbot.

El lector puede desear comparar varios chatterbots y evaluar su inteligencia en el Ejercicio 1.1 en la página 14.

1.2 La historia de la IA

La IA se basa en muchos logros científicos pasados que no se mencionan aquí, ya que la IA como ciencia por derecho propio solo existe desde mediados del siglo XX. La Tabla 1.1 en la página 10, con los hitos más importantes de la IA, y una representación gráfica de los principales movimientos de la IA en la Fig. 1.3 en la página 6 complementan el siguiente texto.

1.2.1 Los primeros comienzos

En la década de 1930, Kurt Gödel, Alonso Church y Alan Turing sentaron importantes bases para la lógica y la informática teórica. De particular interés para la IA son los teoremas de Gödel. El teorema de completitud establece que la lógica de predicados de primer orden es completa. Esto significa que todo enunciado verdadero que pueda formularse en la lógica de predicados es demostrable usando las reglas de un cálculo formal. Sobre esta base, los probadores automáticos de teoremas podrían construirse más tarde como implementaciones de cálculos formales.

Con el teorema de incompletud, Gödel demostró que en las lógicas de orden superior existen enunciados verdaderos que son indemostrables.² Con esto descubrió los dolorosos límites de los sistemas formales.

La prueba de Alan Turing de la indecidibilidad del problema de la detención también cae en este período de tiempo. Demostró que no hay ningún programa que pueda decidir si un programa arbitrario determinado (y su respectiva entrada) se ejecutará en un bucle infinito. Con

² Las lógicas de orden superior son extensiones de la lógica de predicados, en las que no solo las variables, sino también los símbolos de funciones o predicados pueden aparecer como términos en una cuantificación. De hecho, Gödel solo demostró que cualquier sistema que se base en la lógica de predicados y pueda formular la aritmética de Peano es incompleto.

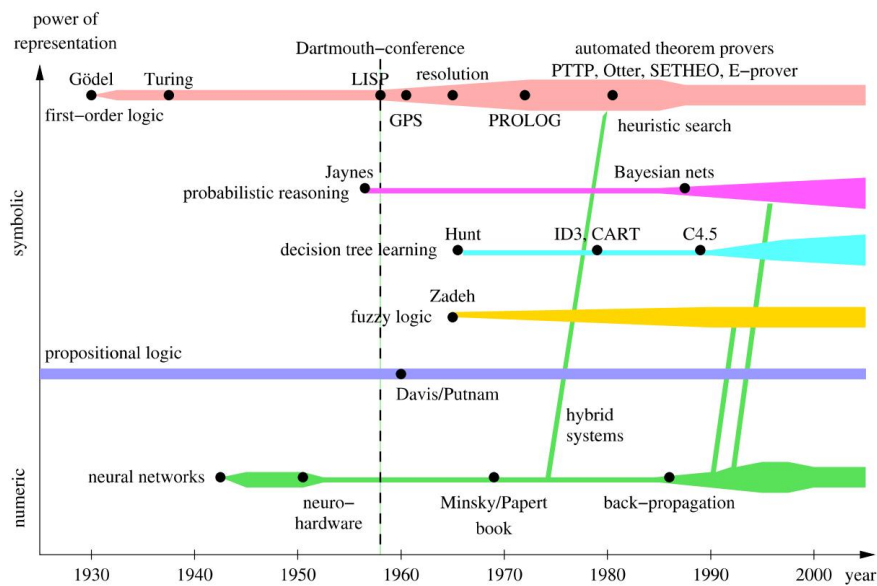


Fig. 1.3 Historia de las distintas áreas de IA. El ancho de las barras indica la prevalencia del uso del método

este Turing también identificó un límite para los programas inteligentes. De ello se deduce, por ejemplo, que nunca habrá un sistema universal de verificación de programas.³ En la

década de 1940, basándose en los resultados de la neurociencia, McCulloch, Pitts y Hebb diseñaron los primeros modelos matemáticos de redes neuronales. Sin embargo, las computadoras en ese momento carecían de la potencia suficiente para simular cerebros simples.

1.2.2 La lógica resuelve (casi) todos los problemas

Por supuesto, la IA como ciencia práctica de la mecanización del pensamiento solo podría comenzar una vez que existieran las computadoras programables. Este fue el caso en la década de 1950. Newell y Simon introdujeron Logic Theorist, el primer probador automático de teoremas y, por lo tanto, también demostraron que con las computadoras, que en realidad solo funcionan con números, también se pueden procesar símbolos. Al mismo tiempo McCarthy introdujo, con el lenguaje LISP, un lenguaje de programación especialmente creado para el procesamiento de estructuras simbólicas. Ambos sistemas se introdujeron en 1956 en la histórica Conferencia de Dartmouth, que se considera el nacimiento de la IA.

En los EE. UU., LISP se convirtió en la herramienta más importante para la implementación de sistemas de IA de procesamiento de símbolos. A partir de entonces, la regla de inferencia lógica conocida como resolución se convirtió en un cálculo completo para la lógica de predicados.

³Esta declaración se aplica a la "corrección total", lo que implica una prueba de ejecución correcta, así como una prueba de finalización para cada entrada válida.

En la década de 1970 se introdujo el lenguaje de programación lógica PROLOG como el Contraparte europea de LISP. PROLOG ofrece la ventaja de permitir la programación utilizando cláusulas de Horn, un subconjunto de la lógica de predicados. Como LISP, PRÓLOGO tiene tipos de datos para el procesamiento conveniente de listas.

Hasta bien entrada la década de 1980, un espíritu innovador dominaba la IA, especialmente entre muchos lógicos. La razón de esto fue la serie de impresionantes logros en procesamiento de símbolos. Con el proyecto de sistemas informáticos de quinta generación en Japón y el programa ESPRIT en Europa, se invirtió mucho en la construcción de computadoras inteligentes.

Para pequeños problemas, probadores automáticos y otros sistemas de procesamiento de símbolos a veces funcionó muy bien. La explosión combinatoria del espacio de búsqueda, sin embargo, definió una ventana muy estrecha para estos éxitos. Esta fase de la IA se describió en [RN10] como "¡Mira, mamá, sin manos!" era.

Debido a que el éxito económico de los sistemas de IA no cumplió con las expectativas, la financiación para la investigación de IA basada en lógica en los Estados Unidos cayó dramáticamente durante la década de 1980.

1.2.3 El nuevo conexionismo

Durante esta fase de desilusión, los informáticos, físicos y Cognitive Los científicos pudieron demostrar, usando computadoras que ahora eran lo suficientemente poderosas, que las redes neuronales modeladas matemáticamente son capaces de aprender usando ejemplos de entrenamiento, para realizar tareas que antes requerían una costosa programación. Debido a la tolerancia a fallas de tales sistemas y su capacidad para reconocer patrones, se hicieron posibles éxitos considerables, especialmente en el reconocimiento de patrones. El reconocimiento facial en fotos y el reconocimiento de escritura a mano son dos ejemplos de aplicaciones. El sistema Nettetalk pudo aprender el habla a partir de textos de ejemplo [SR86]. Bajo el nombre de conexionismo nació una nueva subdisciplina de la IA.

El conexionismo floreció y los subsidios fluyeron. Pero pronto incluso aquí la viabilidad los límites se hicieron evidentes. Las redes neuronales podrían adquirir capacidades impresionantes, pero por lo general no era posible capturar el concepto aprendido en fórmulas simples o reglas lógicas. Los intentos de combinar redes neuronales con reglas lógicas o el conocimiento de expertos humanos se encontró con grandes dificultades. Además, ninguna solución satisfactoria a la estructuración y modularización de las redes.

1.2.4 Razonamiento bajo incertidumbre

La IA como ciencia práctica y orientada a objetivos buscó una salida a esta crisis. Uno deseaba unir la capacidad de la lógica para representar explícitamente el conocimiento con la fuerza de las redes neuronales para manejar la incertidumbre. Se sugirieron varias alternativas.

El razonamiento probabilístico más prometedor trabaja con probabilidades condicionales para fórmulas de cálculo proposicional. Desde entonces se han construido muchos sistemas expertos y de diagnóstico para problemas de razonamiento cotidiano utilizando redes bayesianas.

El éxito de las redes bayesianas radica en su comprensibilidad intuitiva, la

semántica limpia de la probabilidad condicional, y de la teoría de la probabilidad fundada matemáticamente y con siglos de antigüedad.

Las debilidades de la lógica, que sólo puede funcionar con dos valores de verdad, pueden ser resuelto por lógica difusa, que pragmáticamente introduce infinitos valores entre cero y uno. Aunque aún hoy su fundamento teórico no es del todo firme, se está utilizando con éxito, especialmente en ingeniería de control.

Un camino muy diferente condujo a la síntesis exitosa de la lógica y las redes neuronales. bajo el nombre de sistemas híbridos. Por ejemplo, se emplearon redes neuronales para aprender heurísticas para la reducción del enorme espacio de búsqueda combinatoria en el descubrimiento de pruebas [SE90].

Los métodos de aprendizaje de árboles de decisión a partir de datos también funcionan con probabilidades. Los sistemas como CART, ID3 y C4.5 pueden construir de forma rápida y automática muy precisos árboles de decisión que pueden representar conceptos de lógica proposicional y luego ser utilizados como sistemas expertos. Hoy en día son una de las favoritas entre las técnicas de aprendizaje automático. (Secc. 8.4).

Desde aproximadamente 1990, la minería de datos se ha desarrollado como una subdisciplina de la IA en el área. de análisis de datos estadísticos para la extracción de conocimiento de grandes bases de datos. Datos La minería no aporta nuevas técnicas a la IA, sino que introduce el requisito de usar grandes bases de datos para obtener conocimiento explícito. Una aplicación con un gran mercado El potencial está dirigiendo campañas publicitarias de grandes empresas basadas en el análisis de muchos millones de compras realizadas por sus clientes. Por lo general, las técnicas de aprendizaje automático como como el aprendizaje del árbol de decisión entra en juego aquí.

1.2.5 Agentes distribuidos, autónomos y de aprendizaje

La inteligencia artificial distribuida, DAI, ha sido un área de investigación activa desde aproximadamente 1985. Uno de sus objetivos es el uso de computadoras paralelas para aumentar la eficiencia de solucionadores de problemas. Resultó, sin embargo, que debido a la alta computacional complejidad de la mayoría de los problemas, el uso de sistemas "inteligentes" es más beneficioso que la propia paralelización.

Un enfoque conceptual muy diferente resulta del desarrollo de agentes de software autónomos y robots que están destinados a cooperar como equipos humanos. Al igual que con los vehículos Braitenberg antes mencionados, hay muchos casos en los que un agente individual no es capaz de resolver un problema, incluso con recursos ilimitados. Sólo la cooperación de muchos agentes conduce al comportamiento inteligente o a la solución de un problema. Una colonia de hormigas o una colonia de termitas es capaz de erigir edificios de altísima complejidad arquitectónica, a pesar de que ninguna hormiga comprende cómo todo encaja. Esto es similar a la situación de aprovisionando pan para una gran ciudad como Nueva York [RN10]. No existe una agencia central de planificación del pan, sino que hay cientos de panaderos que conocen sus respectivos áreas de la ciudad y hornear la cantidad adecuada de pan en esos lugares.

La adquisición activa de habilidades por parte de los robots es un área apasionante de la investigación actual. Allá hay robots hoy en día, por ejemplo, que aprenden de forma independiente a caminar o a realizar diversas motricidad relacionada con el fútbol (Cap. 10). Aprendizaje cooperativo de múltiples robots para resolver problemas juntos todavía está en pañales.

1.2.6 La IA crece

Los sistemas anteriores que ofrece la IA hoy en día no son una receta universal, sino un taller con un número manejable de herramientas para tareas muy diferentes. La mayoría de estas herramientas están bien desarrolladas y están disponibles como bibliotecas de software terminadas, a menudo con interfaces de usuario convenientes. La selección de la herramienta adecuada y su uso sensato en cada caso individual se deja al desarrollador de IA o al ingeniero de conocimiento. Como cualquier otra artesanía, esto requiere una sólida educación, que este libro pretende promover.

Más que casi cualquier otra ciencia, la IA es interdisciplinaria, ya que se basa en descubrimientos interesantes de campos tan diversos como la lógica, la investigación de operaciones, la estadística, la ingeniería de control, el procesamiento de imágenes, la lingüística, la filosofía, la psicología y la neurobiología. Además de eso, está el área temática de la aplicación en particular. Por lo tanto, desarrollar con éxito un proyecto de IA no siempre es tan simple, pero casi siempre es extremadamente emocionante.

1.3 Agentes

Aunque el término agentes inteligentes no es nuevo para la IA, solo en los últimos años ha ganado protagonismo a través de [RN10], entre otros. Agente denota bastante generalmente un sistema que procesa información y produce una salida a partir de una entrada. Estos agentes pueden clasificarse de muchas maneras diferentes.

En la informática clásica, los agentes de software se emplean principalmente (Fig. 1.4 en la página 11). En este caso, el agente consta de un programa que calcula un resultado a partir de la entrada del usuario.

En robótica, por otro lado, se emplean agentes de hardware (también llamados robots), que además tienen a su disposición sensores y actuadores (Fig. 1.5 en la página 11). El agente puede percibir su entorno con los sensores. Con los actuadores realiza acciones y cambia su entorno.

Con respecto a la inteligencia del agente, existe una distinción entre los agentes reflejos, que solo reaccionan a la entrada, y los agentes con memoria, que también pueden incluir el pasado en sus decisiones. Por ejemplo, un robot conductor que a través de sus sensores conoce su posición exacta (y la hora) no tiene forma, como agente reflejo, de determinar su velocidad. Sin embargo, si guarda la posición, en pasos de tiempo cortos y discretos, puede calcular fácilmente su velocidad promedio en el intervalo de tiempo anterior.

Si un agente reflejo está controlado por un programa determinista, representa una función del conjunto de todas las entradas al conjunto de todas las salidas. Un agente con memoria, por otro lado, en general no es una función. ¿Por qué? (Consulte el ejercicio 1.5 en la página 14). Los agentes reflejos son suficientes en los casos en que el problema a resolver implica un proceso de decisión de Markov. Este es un proceso en el que solo se necesita el estado actual para determinar la próxima acción óptima (consulte el Capítulo 10).

Un robot móvil que debe moverse de la habitación 112 a la habitación 179 en un edificio realiza acciones diferentes a las de un robot que debe moverse a la habitación 105. En otras palabras, las acciones dependen del objetivo. Dichos agentes se denominan basados en objetivos.

Tabla 1.1 Hitos en el desarrollo de la IA desde Gödel hasta hoy 1931 El austriaco Kurt Gödel

muestra que en la lógica de predicados de primer orden todos los enunciados verdaderos son derivables [Göd31a]. En la lógica de orden superior, por otro lado, hay declaraciones verdaderas que no son demostrables [Göd31b]. (En [Göd31b] Gödel mostró que la lógica de predicados extendida con los axiomas de la aritmética es incompleta).
1937 Alan Turing señala los límites de las máquinas inteligentes con el problema de la detención [Tur37].
1943 McCulloch y Pitts modelan redes neuronales y establecen la conexión con proposiciones lógicas.
1950 Alan Turing define la inteligencia artificial con la prueba de Turing y escribe sobre máquinas de aprendizaje y algoritmos genéticos [Tur50].
1951 Marvin Minsky desarrolla una máquina de redes neuronales. Con 3000 tubos de vacío él simula 40 neuronas.
1955 Arthur Samuel (IBM) construye un programa de aprendizaje de ajedrez que juega mejor que su desarrollador [Sam59].
1956 McCarthy organiza una conferencia en Dartmouth College. Aquí el nombre Artificial La inteligencia se introdujo por primera vez. Newell y Simon de la Universidad Carnegie Mellon (CMU) presentan Logic Theorist, el primer programa informático de procesamiento de símbolos [NSS83].
1958 McCarthy inventa en el MIT (Massachusetts Institute of Technology) el lenguaje de alto nivel LISP. Escribe programas que son capaces de modificarse a sí mismos.
1959 Gelernter (IBM) construye el probador de teoremas de geometría.
1961 El Solucionador General de Problemas (GPS) de Newell y Simon imita el pensamiento humano [NS61].
1963 McCarthy funda el Laboratorio de IA en la Universidad de Stanford.
1965 Robinson inventa el cálculo de resolución para la lógica de predicados [Rob65] (Sección 3.5).
1966 El programa de Weizenbaum Eliza lleva a cabo diálogos con personas en lenguaje natural [Wei66] (Sect. 1.1.2).
1969 Minsky y Papert muestran en su libro Perceptrons que el perceptrón, un muy simple red neuronal, solo puede representar funciones lineales [MP69] (Apt. 1.1.2).
1972 El científico francés Alain Colmerauer inventa el lenguaje de programación lógica PROLOG (Cap. 5). El médico británico de Dombal desarrolla un sistema experto para el diagnóstico del dolor abdominal agudo [dDLS+72]. Pasa desapercibido en la comunidad de IA dominante de la época (Sect. 7.3).
1976 Shortliffe y Buchanan desarrollan MYCIN, un sistema experto para el diagnóstico de enfermedades infecciosas, que es capaz de manejar la incertidumbre (Cap. 7).
1981 Japón inicia, a un gran costo, el "Proyecto de Quinta Generación" con el objetivo de construyendo una potente máquina PROLOG.
1982 R1, el sistema experto para configurar computadoras, ahorra a Digital Equipment Corporation 40 millones de dólares al año [McD82].
1986 Renacimiento de las redes neuronales a través, entre otros, de Rumelhart, Hinton y Sejnowski [RM86]. El sistema Nottalk aprende a leer textos en voz alta [SR86] (Cap. 9).
1990 Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter incorporan la teoría de la probabilidad a la IA con redes bayesianas (Sección 7.4). Los sistemas multiagente se vuelven populares.
1992 El programa Tesauros TD-gammon demuestra las ventajas del refuerzo aprendiendo.
1993 Iniciativa mundial RoboCup para construir robots autónomos que juegan al fútbol [Roba].

Tabla 1.1 (continuación)

1995	A partir de la teoría del aprendizaje estadístico, Vapnik desarrolla máquinas de vectores de soporte, que son muy importantes en la actualidad.
1997	La computadora de ajedrez de IBM Deep Blue derrota al campeón mundial de ajedrez Gary Kasparov. Primera competencia internacional de RoboCup en Japón.
2003	Los robots en RoboCup demuestran de manera impresionante lo que la IA y la robótica son capaces de hacer logrando
2006	La robótica de servicios se convierte en una importante área de investigación de IA.
2010	Los robots autónomos comienzan a aprender sus políticas.
2011	El programa de comprensión del lenguaje natural y respuesta a preguntas de IBM "Watson" derrota a dos campeones humanos en el programa de concursos de la televisión estadounidense "Jeopardy!" (Secc. 1.4).

Fig. 1.4 Un agente de software con interacción con el usuario

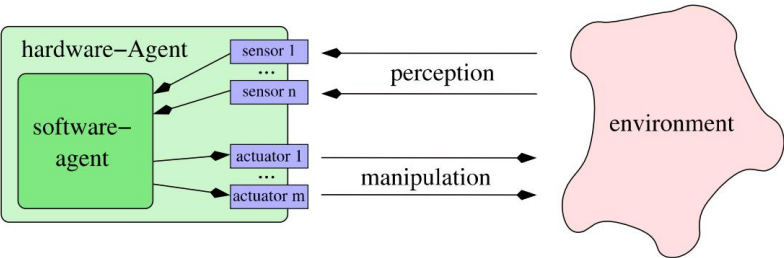
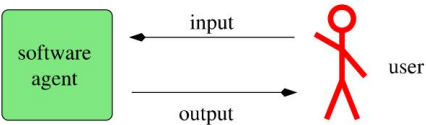


Fig. 1.5 Un agente de hardware

Ejemplo 1.1 Un filtro de spam es un agente que clasifica los correos electrónicos entrantes en categorías deseadas o no deseadas (spam) y elimina los correos electrónicos no deseados. Su objetivo como agente basado en objetivos es poner todos los correos electrónicos en la categoría correcta. En el curso de esta tarea no tan simple, el agente puede ocasionalmente cometer errores. Debido a que su objetivo es clasificar todos los correos electrónicos correctamente, intentará cometer la menor cantidad de errores posible. Sin embargo, eso no es siempre lo que el usuario tiene en mente. Comparemos los siguientes dos agentes. De 1000 correos electrónicos, el Agente 1 comete solo 12 errores. El agente 2, por otro lado, comete 38 errores con los mismos 1000 correos electrónicos. ¿Es por lo tanto peor que el Agente 1? Los errores de ambos agentes se muestran con más detalle en la siguiente tabla, la denominada "matriz de confusión":

Agente 1:				Agente 2:			
		clase correcta				clase correcta	
		buscado spam 1				buscado spam	
filtro de spam decide	buscado 189			filtro de spam decide	buscado 200 38 spam 0		
	spam 11 799				762		

De hecho, el Agente 1 comete menos errores que el Agente 2, pero esos pocos errores son graves porque el usuario pierde 11 correos electrónicos potencialmente importantes. Debido a que en este caso hay dos tipos de errores de diferente gravedad, cada error debe ponderarse con el factor de costo apropiado (consulte la Sección 7.3.5 y el Ejercicio 1.7 en la página 14).

La suma de todos los errores ponderados da el costo total causado por decisiones erróneas. El objetivo de un agente basado en costos es minimizar el costo de las decisiones erróneas a largo plazo, es decir, en promedio. Insecto. 7.3 nos familiarizaremos con el sistema de diagnóstico LEXMED como ejemplo de un agente basado en costos.

Análogamente, el objetivo de un agente basado en la utilidad es maximizar la utilidad derivada de las decisiones correctas a largo plazo, es decir, en promedio. La suma de todas las decisiones ponderadas por sus respectivos factores de utilidad da la utilidad total.

De particular interés en IA son los agentes de aprendizaje, que son capaces de cambiarse a sí mismos dados ejemplos de entrenamiento o mediante retroalimentación positiva o negativa, de modo que la utilidad promedio de sus acciones crece con el tiempo (ver Capítulo 8).

Como se menciona en la Secc. 1.2.5, los agentes distribuidos se utilizan cada vez más, cuya inteligencia no está localizada en un agente, sino que solo se puede ver a través de la cooperación de muchos agentes.

El diseño de un agente está orientado, junto con su objetivo, fuertemente hacia su entorno, o alternativamente su imagen del entorno, que depende fuertemente de sus sensores. El entorno es observable si el agente siempre conoce el estado completo del mundo. De lo contrario, el entorno es sólo parcialmente observable.

Si una acción siempre conduce al mismo resultado, entonces el entorno es determinista.

De lo contrario, es no determinista. En un entorno discreto, solo ocurren un número finito de estados y acciones, mientras que un entorno continuo cuenta con un número infinito de estados o acciones.

1.4 Sistemas basados en el conocimiento Un

agente es un programa que implementa un mapeo desde las percepciones hasta las acciones. Para los agentes simples, esta forma de ver el problema es suficiente. Para aplicaciones complejas en las que el agente debe poder confiar en una gran cantidad de información y debe realizar una tarea difícil, la programación del agente puede ser muy costosa y poco clara sobre cómo proceder. Aquí Al proporciona un camino claro a seguir que simplificará enormemente el trabajo.

En primer lugar, separamos el conocimiento del sistema o programa, que utiliza el conocimiento para, por ejemplo, llegar a conclusiones, responder consultas o idear un plan.

Este sistema se denomina mecanismo de inferencia. El conocimiento se almacena en una base de conocimientos (KB). La adquisición de conocimiento en la base de conocimiento se denomina Ingeniería de conocimiento y se basa en varias fuentes de conocimiento, como expertos humanos, el ingeniero de conocimiento y bases de datos. Los sistemas de aprendizaje activo también pueden adquirir conocimiento a través de la exploración activa del mundo (ver Capítulo 10). En la Fig. 1.6 en la página 13 se presenta la arquitectura general de los sistemas basados en el conocimiento.

Avanzar hacia una separación de conocimiento e inferencia tiene varias ventajas cruciales. La separación del conocimiento y la inferencia puede permitir que los sistemas de inferencia se implementen de una manera en gran medida independiente de la aplicación. Por ejemplo, es mucho