

LEHRBUCH

Wolfgang Ertel

Grundkurs Künstliche Intelligenz

Eine praxisorientierte Einführung

3. Auflage



Springer Vieweg

Grundkurs Künstliche Intelligenz

Wolfgang Ertel

Grundkurs Künstliche Intelligenz

Eine praxisorientierte Einführung



Springer Vieweg

Wolfgang Ertel
Hochschule Ravensburg-Weingarten
Weingarten, Deutschland

ISBN 978-3-8348-1677-1
DOI 10.1007/978-3-8348-2157-7

ISBN 978-3-8348-2157-7 (eBook)

Die Deutsche Nationalbibliothek verzeichnetet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Vieweg
© Springer Fachmedien Wiesbaden 2008, 2009, 2013
Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media
www.springer-vieweg.de

Vorwort zur dritten Auflage

In Kap. 8 gibt es nun zwei neue Abschnitte. Das wichtige Thema Kreuzvalidierung erhält in Abschn. 8.5 endlich den verdienten Raum. In Abschn. 8.8 über One-Class-Learning werden Verfahren zum Lernen von Klassifikatoren beschrieben, wenn nur Daten einer Klasse verfügbar sind.

Neben der Fehlerkorrektur wurden an vielen Stellen Aktualisierungen vorgenommen sowie die Literaturliste ergänzt um wichtige neue Standardwerke wie etwa [Bis06]. Die Geschichte der KI wurde ergänzt um die ganz wichtige aktuelle Strömung hin zu autonomen Maschinen und Robotern. Kapitel 6 wurde ergänzt durch neue Trends bei Schachcomputern. Interessant ist, dass die neuesten Entwicklungen beim Schach nicht mehr auf Hardwaredleistung, sondern auf lernfähige KI setzen. Die Nearest Neighbour-Methoden in Kap. 8 wurden ergänzt durch ein anschauliches Beispiel.

Ravensburg, März 2013

Wolfgang Ertel

Vorwort zur zweiten Auflage

Ich bedanke mich bei allen Lesern des Buches und ganz besonders bei Richard Cubek für Kommentare, Kritik und die Meldung von Fehlern. Neben der Fehlerkorrektur waren an einigen Stellen Ergänzungen und Änderungen angebracht. So überarbeitete ich zum Beispiel die Beschreibung der Normalformtransformation für prädikatenlogische Formeln und fügte beim Perzeptron ergänzende Erklärungen ein. Die Beschreibung des Backpropagation-Algorithmus wurde durch Pseudocode verbessert und die Einführung in das Clustering ergänzte ich zur Veranschaulichung durch ein Bild. Das Kapitel zehn zum Lernen durch Verstärkung wurde erweitert um einige wichtige Strömungen in der aktuellen Forschung und zusätzliche Anwendungen. Ich hoffe, dadurch die Faszination der künstlichen Intelligenz noch ein wenig besser vermitteln zu können.

Ravensburg, Mai 2009

Wolfgang Ertel

Vorwort zur ersten Auflage

Mit dem Verstehen von Intelligenz und dem Bau intelligenter Systeme gibt sich die Künstliche Intelligenz (KI) ein Ziel vor. Die auf dem Weg zu diesem Ziel zu verwendenden Methoden und Formalismen sind aber nicht festgelegt, was dazu geführt hat, dass die KI heute aus einer Vielzahl von Teildisziplinen besteht. Die Schwierigkeit bei einem KI-Grundkurs liegt darin, einen Überblick über möglichst alle Teilgebiete zu vermitteln, ohne allzu viel Verlust an Tiefe und Exaktheit.

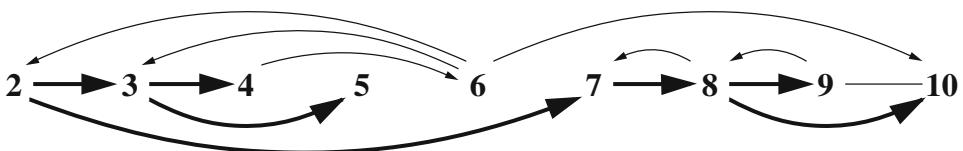
Das Buch von Russell und Norvig [RN03] definiert heute quasi den Standard zur Einführung in die KI. Da dieses Buch aber mit 1327 Seiten in der deutschen Ausgabe für die meisten Studierenden zu umfangreich und zu teuer ist, waren die Vorgaben für das zu schreibende Buch klar: Es sollte eine für Studierende erschwingliche Einführung in die moderne KI zum Selbststudium oder als Grundlage für eine vierstündige Vorlesung mit maximal 300 Seiten werden. Das Ergebnis liegt nun hier vor.

Bei einem Umfang von ca. 300 Seiten kann ein dermaßen umfangreiches Gebiet wie die KI nicht vollständig behandelt werden. Damit das Buch nicht zu einer Inhaltsangabe wird, habe ich versucht, in jedem der Teilgebiete Agenten, Logik, Suche, Schließen mit Unsicherheit, maschinelles Lernen und Neuronale Netze an einigen Stellen etwas in die Tiefe zu gehen und konkrete Algorithmen und Anwendungen vorzustellen.

Nicht im Detail behandelt werden die Gebiete Bildverarbeitung, Fuzzy-Logik, und die Verarbeitung natürlicher Sprache. Das für die gesamte Informatik wichtige Gebiet der Bildverarbeitung stellt eine eigenständige Disziplin mit sehr guten Lehrbüchern, zum Beispiel [Jäh05] dar. Einen ähnlichen Status hat die Verarbeitung natürlicher Sprache. Beim Erkennen und auch beim Erzeugen von Texten und gesprochener Sprache kommen unter anderem Methoden aus der Logik, dem probabilistischen Schließen sowie neuronale Netze zur Anwendung. Insofern gehört dieses Gebiet zur KI. Andererseits ist auch die Computerlinguistik ein eigenes umfangreiches Teilgebiet der Informatik mit vielen Gemeinsamkeiten zu formalen Sprachen. Wir werden in diesem Buch an einigen Stellen auf entsprechende Systeme hinweisen, aber keine systematische Einführung bereitstellen. Für eine erste Einführung in dieses Gebiet verweisen wir auf die Kapitel 22 und 23 in [RN03]. Die Fuzzy-Logik, beziehungsweise die Fuzzy-Mengentheorie hat sich aufgrund ihrer primären Anwendungen in der Automatisierungstechnik zu einem Teilgebiet der Regelungstechnik

entwickelt und wird auch in entsprechenden Büchern und Vorlesungen behandelt. Daher verzichten wir hier auf eine Einführung.

In dem unten dargestellten Graphen sind die Abhängigkeiten der Kapitel des Buches grob skizziert. Um die Darstellung übersichtlich zu halten, wurde Kap. 1 mit der für alle weiteren Kapitel grundlegenden Einführung nicht eingezeichnet. Ein dicker Pfeil von Kap. 2 nach 3 zum Beispiel bedeutet, dass die Aussagenlogik für das Verständnis der Prädikatenlogik vorausgesetzt wird. Der dünne Pfeil von Kap. 9 nach 10 bedeutet, dass Neuronale Netze für das Verständnis des Lernens durch Verstärkung hilfreich sind, aber nicht dringend benötigt werden. Dünne Rückwärtspfeile sollen deutlich machen, dass spätere Kapitel das Verständnis schon gelernter Themen vertiefen können.



Das Buch wendet sich an Studierende der Informatik und anderer technisch naturwissenschaftlicher Fächer und setzt überwiegend nur Mathematikkenntnisse der Oberstufe voraus. An einigen Stellen werden Kenntnisse aus der linearen Algebra und der mehrdimensionalen Analysis benötigt. Zu einem tieferen Verständnis der Inhalte ist die aktive Beschäftigung mit den Übungen unerlässlich. Das bedeutet, dass die Musterlösung nur nach intensiver Beschäftigung mit der jeweiligen Aufgabe zur Kontrolle konsultiert werden sollte, getreu dem Motto „*Studium ohne Hingabe schadet dem Gehirn*“ von Leonardo da Vinci. Etwas schwierigere Aufgaben sind mit *, besonders schwierige mit ** markiert. Aufgaben, die Programmier- oder spezielle Informatikkenntnisse erfordern, sind mit => gekennzeichnet.

Auf der Webseite zum Buch unter <http://www.hs-weingarten.de/~ertel/kibuch> sind digitale Materialien zu den Übungen wie zum Beispiel Trainingsdaten für Lernalgorithmen, eine Seite mit Verweisen auf die im Buch erwähnten KI-Programme, eine Liste mit Links zu den behandelten Themen, eine anklickbare Liste der Literaturverweise, eine Errataliste und Präsentationsfolien für Dozenten zu finden. Ich möchte den Leser bitten, Anregungen, Kritik und Hinweise auf Fehler direkt an ertel@hs-weingarten.de zu senden.

Bedanken möchte ich mich an erster Stelle bei meiner Frau Evelyn, die mir den Rücken frei hielt für das Schreiben. Ein besonderer Dank geht an Wolfgang Bibel und an Chris Lobenschuss, die das Manuskript sehr sorgfältig korrigierten. Ihre Anmerkungen und Diskussionen führten zu vielen Verbesserungen und Ergänzungen. Für das Korrekturlesen und andere wertvolle Dienste möchte ich mich bedanken bei Celal Döven, Joachim Feßler, Nico Hochgeschwender, Paul Kirner, Wilfried Meister, Norbert Perk, Peter Radtke, Markus Schneider, Manfred Schramm, Uli Stärk, Michel Tokic, Arne Usadel und allen interessierten Studierenden. Mein Dank geht auch an Florian Mast für die überaus gelungenen

Cartoons und die sehr effektive Zusammenarbeit. Für die kooperative und fruchtbare Zusammenarbeit mit dem Vieweg Verlag bedanke ich mich bei Günter Schulz und Sybille Thelen.

Ravensburg, September 2007

Wolfgang Ertel

Inhaltsverzeichnis

1	Einführung	1
1.1	Was ist Künstliche Intelligenz	1
1.2	Geschichte der KI	6
1.3	Agenten	12
1.4	Wissensbasierte Systeme	14
1.5	Übungen	16
2	Aussagenlogik	19
2.1	Syntax	19
2.2	Semantik	20
2.3	Beweisverfahren	23
2.4	Resolution	27
2.5	Hornklauseln	30
2.6	Berechenbarkeit und Komplexität	33
2.7	Anwendungen und Grenzen	34
2.8	Übungen	34
3	Prädikatenlogik erster Stufe	37
3.1	Syntax	38
3.2	Semantik	39
3.3	Quantoren und Normalformen	44
3.4	Beweiskalküle	47
3.5	Resolution	49
3.6	Automatische Theorembeweiser	54
3.7	Mathematische Beispiele	56
3.8	Anwendungen	59
3.9	Zusammenfassung	61
3.10	Übungen	62
4	Grenzen der Logik	65
4.1	Das Suchraumproblem	65
4.2	Entscheidbarkeit und Unvollständigkeit	67

4.3	Der fliegende Pinguin	69
4.4	Modellierung von Unsicherheit	72
4.5	Übungen	73
5	Logikprogrammierung mit Prolog	75
5.1	Prolog-Systeme und Implementierungen	76
5.2	Einfache Beispiele	76
5.3	Ablaufsteuerung und prozedurale Elemente	79
5.4	Listen	81
5.5	Selbstmodifizierende Programme	83
5.6	Ein Planungsbeispiel	84
5.7	Constraint Logic Programming	86
5.8	Zusammenfassung	88
5.9	Übungen	88
6	Suchen, Spielen und Probleme lösen	93
6.1	Einführung	93
6.2	Uninformierte Suche	100
6.3	Heuristische Suche	105
6.4	Spiele mit Gegner	114
6.5	Heuristische Bewertungsfunktionen	118
6.6	Stand der Forschung	120
6.7	Übungen	122
7	Schließen mit Unsicherheit	125
7.1	Rechnen mit Wahrscheinlichkeiten	127
7.2	Die Methode der Maximalen Entropie	135
7.3	LEXMED, ein Expertensystem für Appendizitisdiagnose	144
7.4	Schließen mit Bayes-Netzen	158
7.5	Zusammenfassung	171
7.6	Übungen	172
8	Maschinelles Lernen und Data Mining	177
8.1	Datenanalyse	183
8.2	Das Perzeptron, ein linearer Klassifizierer	185
8.3	Nearest Neighbour-Methoden	192
8.4	Lernen von Entscheidungsbäumen	202
8.5	Kreuzvalidierung und Überanpassung	218
8.6	Lernen von Bayes-Netzen	220
8.7	Der Naive-Bayes-Klassifizierer	223
8.8	One-Class-Learning	228
8.9	Clustering	230
8.10	Data Mining in der Praxis	235

8.11	Zusammenfassung	239
8.12	Übungen	241
9	Neuronale Netze	247
9.1	Von der Biologie zur Simulation	248
9.2	Hopfield-Netze	253
9.3	Neuronale Assoziativspeicher	259
9.4	Lineare Netze mit minimalem Fehler	268
9.5	Der Backpropagation-Algorithmus	274
9.6	Support-Vektor-Maschinen	281
9.7	Anwendungen	282
9.8	Zusammenfassung und Ausblick	283
9.9	Übungen	284
10	Lernen durch Verstärkung (Reinforcement Learning)	287
10.1	Einführung	287
10.2	Die Aufgabenstellung	290
10.3	Uninformierte kombinatorische Suche	291
10.4	Wert-Iteration und Dynamische Programmierung	293
10.5	Ein lernender Laufroboter und seine Simulation	296
10.6	Q-Lernen	298
10.7	Erkunden und Verwerten	302
10.8	Approximation, Generalisierung und Konvergenz	303
10.9	Anwendungen	304
10.10	Fluch der Dimensionen	305
10.11	Zusammenfassung und Ausblick	306
10.12	Übungen	307
11	Lösungen zu den Übungen	309
11.1	Einführung	309
11.2	Aussagenlogik	310
11.3	Prädikatenlogik	312
11.4	Grenzen der Logik	314
11.5	Prolog	314
11.6	Suchen, Spielen und Probleme lösen	316
11.7	Schließen mit Unsicherheit	319
11.8	Maschinelles Lernen und Data Mining	325
11.9	Neuronale Netze	333
11.10	Lernen durch Verstärkung	334
Literatur	339	
Sachverzeichnis	347	

1.1 Was ist Künstliche Intelligenz

Der Begriff *Künstliche Intelligenz* weckt Emotionen. Zum einen ist da die Faszination der *Intelligenz*, die offenbar uns Menschen eine besondere Stellung unter den Lebewesen verleiht. Es stellen sich Fragen wie „Was ist Intelligenz?“, „Wie kann man Intelligenz messen?“ oder „Wie funktioniert unser Gehirn?“. All diese Fragen sind von Bedeutung für das Verständnis von künstlicher Intelligenz. Die zentrale Frage für den Ingenieur, speziell für den Informatiker, ist jedoch die Frage nach der intelligenten Maschine, die sich verhält wie ein Mensch, die intelligentes Verhalten zeigt.

Das Attribut *künstlich* weckt eventuell ganz andere Assoziationen. Da kommen Ängste vor intelligenten Robotermenschen auf. Es werden Bilder aus Science-Fiction-Romanen wach. Es stellt sich die Frage, ob wir überhaupt versuchen sollten, unser höchstes Gut, den Geist, zu verstehen, zu modellieren oder gar nachzubauen.

Bei derart unterschiedlichen Interpretationen schon auf den ersten Blick wird es schwierig, den Begriff **Künstliche Intelligenz** oder **KI** (engl. *artificial intelligence* oder *AI*) einfach und prägnant zu definieren. Trotzdem möchte ich versuchen, anhand von Beispielen und historischen Definitionen das Gebiet der KI zu charakterisieren. John McCarthy, einer der Pioniere der KI, definierte 1955 als Erster den Begriff *Künstliche Intelligenz* etwa so:

Ziel der KI ist es, Maschinen zu entwickeln, die sich verhalten, als verfügten sie über Intelligenz.

Um diese Definition zu testen, möge sich der Leser folgendes Szenario vorstellen. Auf einer abgegrenzten vier mal vier Meter großen Fläche bewegen sich etwa fünfzehn kleine Roboterfahrzeuge. Man kann verschiedene Verhaltensweisen beobachten. Die einen formen kleine Grüppchen mit relativ wenig Bewegung. Andere bewegen sich ruhig durch den Raum und weichen jeder Kollision elegant aus. Wieder andere folgen anscheinend einem Führer. Auch aggressives Verhalten kann bei einigen beobachtet werden. Sehen wir hier intelligentes Verhalten?

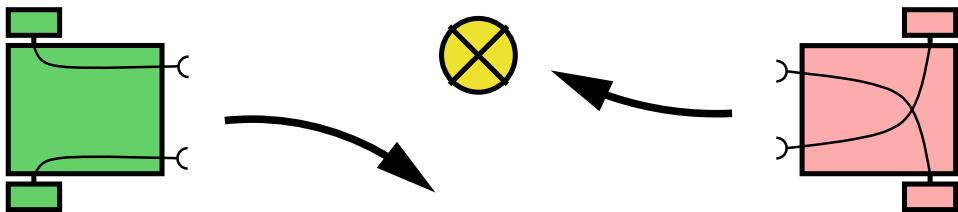


Abb. 1.1 Zwei ganz einfache Braitenberg-Vehikel und deren Reaktion auf eine Lichtquelle

Nach der Definition von McCarthy könnten die eben erwähnten Roboter als intelligent bezeichnet werden. Der Psychologe Valentin Braitenberg hat gezeigt, dass sich dieses scheinbar komplexe Verhalten mit ganz einfachen elektrischen Verschaltungen erzeugen lässt [Bra84]. Die so genannten Braitenberg-Vehikel haben zwei Räder, von denen jedes einzelne durch einen Elektromotor angetrieben wird. Die Geschwindigkeit jedes Motors wird beeinflusst durch einen Lichtsensor an der Vorderseite des Vehikels wie in Abb. 1.1 dargestellt. Je mehr Licht auf den Sensor trifft, desto schneller läuft der Motor. Das Vehikel 1 in der Abbildung links wird sich aufgrund der Verschaltung von einer punktförmigen Lichtquelle weg bewegen. Vehikel 2 hingegen wird sich auf eine Lichtquelle zu bewegen. Mit weiteren kleinen Modifikationen können noch andere Verhaltensweisen erreicht werden, so dass mit diesen ganz einfachen Vehikeln das oben beschriebene beeindruckende Verhalten erzielt werden kann.

Offenbar ist die obige Definition nicht ausreichend, denn die KI setzt sich zum Ziel, viele schwierige praktische Probleme zu lösen, womit die Braitenberg-Vehikel sicher überfordert wären. In der Encyclopedia Britannica [Bri91] findet man eine Definition, die in etwa so lautet:

KI ist die Fähigkeit digitaler Computer oder computergesteuerter Roboter, Aufgaben zu lösen, die normalerweise mit den höheren intellektuellen Verarbeitungsfähigkeiten von Menschen in Verbindung gebracht werden ...

Aber auch diese Definition hat noch ihre Schwächen. Sie würde zum Beispiel einem Computer, der einen langen Text speichern und jederzeit abrufen kann, intelligente Fähigkeiten zugestehen, denn das Auswendiglernen langer Texte kann durchaus als *höhere intellektuelle Verarbeitungsfähigkeit* von Menschen verstanden werden, genauso wie auch zum Beispiel das schnelle Multiplizieren von zwei zwanzistelligen Zahlen. Nach dieser Definition ist also jeder Computer ein KI-System. Dieses Dilemma wird elegant gelöst durch die folgende Definition von Elaine Rich [Ric83]:

Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

Knapp und prägnant charakterisiert Rich das, was die Wissenschaftler in der KI seit etwa fünfzig Jahren tun. Auch im Jahr 2050 wird diese Definition immer noch aktuell sein.

Aufgaben wie etwa das Ausführen von vielen Berechnungen in kurzer Zeit sind die Stärken digitaler Computer. Sie leisten hier um ein Vielfaches mehr als jeder Mensch. In vielen anderen Bereichen sind wir Menschen jedoch den Maschinen weit überlegen. Betritt etwa ein Mensch einen ihm unbekannten Raum, so kann er binnen Sekundenbruchteilen die Szene erkennen und, falls nötig, genauso schnell Entscheidungen treffen und Aktionen planen. Autonome¹ Roboter sind mit dieser Aufgabe heute noch überfordert. Nach der Definition von Rich ist das also eine Aufgabe der KI. Tatsächlich ist heute in der KI die Forschung an autonomen Robotern ein wichtiges aktuelles Thema. Der Bau von Schachcomputern hingegen hat an Bedeutung verloren, denn diese spielen schon auf oder über dem Niveau von Weltmeistern.

Gefährlich wäre es jedoch, aus der Definition von Rich den Schluss zu ziehen, dass die KI sich nur mit der pragmatischen praktischen Implementierung intelligenter Verfahren beschäftigt. Intelligente Systeme im Sinne der Definition von Rich kann man nicht bauen ohne ein tiefes Verständnis des menschlichen Schließens und intelligenten Handelns generell, weshalb zum Beispiel die Kognitionswissenschaft (siehe Abschn. 1.1.1) für die KI von großer Wichtigkeit ist. Dies zeigt auch, dass die anderen erwähnten Definitionen wichtige Aspekte der KI wiedergeben.

Eine besondere Stärke menschlicher Intelligenz ist die Adaptivität. Wir sind in der Lage, uns an die verschiedensten Umweltbedingungen anzupassen und durch **Lernen** unser Verhalten entsprechend zu ändern. Da wir gerade in der Lernfähigkeit den Computern noch weit überlegen sind, ist nach der Definition von Rich das **maschinelle Lernen** ein zentrales Teilgebiet der KI.

1.1.1 Hirnforschung und Problemlösen

Beim Erforschen intelligenter Verfahren kann man versuchen zu verstehen, wie das menschliche Gehirn arbeitet und dieses dann auf dem Computer zu modellieren oder zu simulieren. Viele Ideen und Prinzipien im Gebiet der neuronalen Netze (siehe Kap. 9) entstammen der Hirnforschung.

Ein ganz anderer Zugang ergibt sich bei einer zielorientierten Vorgehensweise, die vom Problem ausgeht und nach einem möglichst optimalen Lösungsverfahren sucht. Hierbei ist es unwichtig, wie wir Menschen dieses Problem lösen. Die Methode ist bei dieser Vorgehensweise zweitrangig. An erster Stelle steht die optimale intelligente Lösung des Problems. Der Fixpunkt in der KI ist daher meist nicht eine Methode, wie zum Beispiel die Prädikatenlogik, sondern das Ziel, intelligente Agenten für die verschiedensten Aufgaben zu bauen. Da die Aufgaben aber sehr unterschiedlich sein können, ist es nicht erstaunlich, dass die in der KI heute verwendeten Methoden teilweise auch sehr unterschiedlich sind. Ähnlich wie in der Medizin, die viele unterschiedliche, oft lebensrettende Diagnose- und

¹ Ein autonomer Roboter arbeitet selbstständig, ohne manuelle Unterstützung, also insbesondere ohne Fernsteuerung.

Therapieverfahren umfasst, stellt auch die KI heute eine breite Palette an effektiven Verfahren für die verschiedensten Anwendungen bereit. Der Leser möge sich von Abb. 1.2 inspirieren lassen. Genau wie in der Medizin gibt es auch in der KI keine universelle Methode für alle Anwendungsbereiche, aber eine große Zahl möglicher Behandlungen für die verschiedensten großen und kleinen Probleme des Alltags.

Der Erforschung des menschlichen Denkens auf etwas höherer Ebene widmet sich die **Kognitionswissenschaft** (engl. cognitive science). Wie auch die Hirnforschung liefert dieses Gebiet viele wichtige Ideen für die praktische KI. Umgekehrt liefern die Algorithmen und Implementierungen wiederum wichtige Rückschlüsse auf die Funktionsweise des menschlichen Schließens. So stehen diese drei Gebiete in einer fruchtbaren interdisziplinären Wechselwirkung. Gegenstand dieses Buches ist jedoch überwiegend die problemorientierte KI als Teildisziplin der Informatik.

Im Umfeld von Intelligenz und Künstlicher Intelligenz gibt es viele interessante philosophische Fragen. Wir Menschen haben ein Bewusstsein. Das heißt, wir können über uns selbst nachdenken oder sogar darüber nachdenken, dass wir über uns selbst nachdenken. Wie kommt dieses Bewusstsein zustande? Viele Philosophen und Neurologen glauben heute, dass die Seele und das Bewusstsein an die Materie, das heißt, an das Gehirn geknüpft sind. Damit könnte die Frage, ob eine Maschine Seele oder Bewusstsein haben kann, in der Zukunft irgendwann relevant werden. Beim Leib-Seele-Problem zum Beispiel geht es darum, ob die Seele an den Körper gebunden ist oder nicht. Wir werden diese Fragen hier nicht diskutieren. Der interessierte Leser wird verwiesen auf [Spe03, Spe04] und aufgefordert, sich während des Studiums der KI-Techniken über diese Fragen selbst eine Meinung zu bilden.

1.1.2 Der Turing-Test und Chatterbots

Alan Turing hat sich durch seine Definition einer intelligenten Maschine auch als früher Pionier in der KI einen Namen gemacht. Dazu muss die fragliche Maschine folgenden Test bestehen. Die Testperson Alice sitzt in einem abgeschlossenen Raum mit zwei Computerterminals. Ein Terminal ist mit der Maschine verbunden, das andere mit der gutwilligen Person Bob. Alice kann nun an beiden Terminals Fragen eintippen. Sie hat die Aufgabe, nach fünf Minuten zu entscheiden, an welchem Terminal die Maschine antwortet. Die Maschine besteht den Test, wenn sie Alice in mindestens 30 % der Fälle täuschen kann [Tur50].

Dieser Test ist philosophisch sehr interessant. Für die praktische KI, die sich mit der Problemlösung beschäftigt, ist er als Test jedoch wenig relevant. Die Gründe sind ähnlich wie bei den oben erwähnten Braitenberg-Vehikeln (siehe Aufgabe 1.3).

Der KI-Pionier und Gesellschaftskritiker Joseph Weizenbaum hat als erster ein Programm mit dem Namen **Eliza** entwickelt, welches auf eine Testperson wie ein menschlicher Psychologe antworten sollte [Wei66]. Er konnte zeigen, dass dies tatsächlich in vielen Fällen erfolgreich war. Angeblich hat sich seine Sekretärin oft und lang mit diesem Programm unterhalten. Heute gibt es im Netz viele so genannte **Chatterbots**, die teilweise bei den ersten

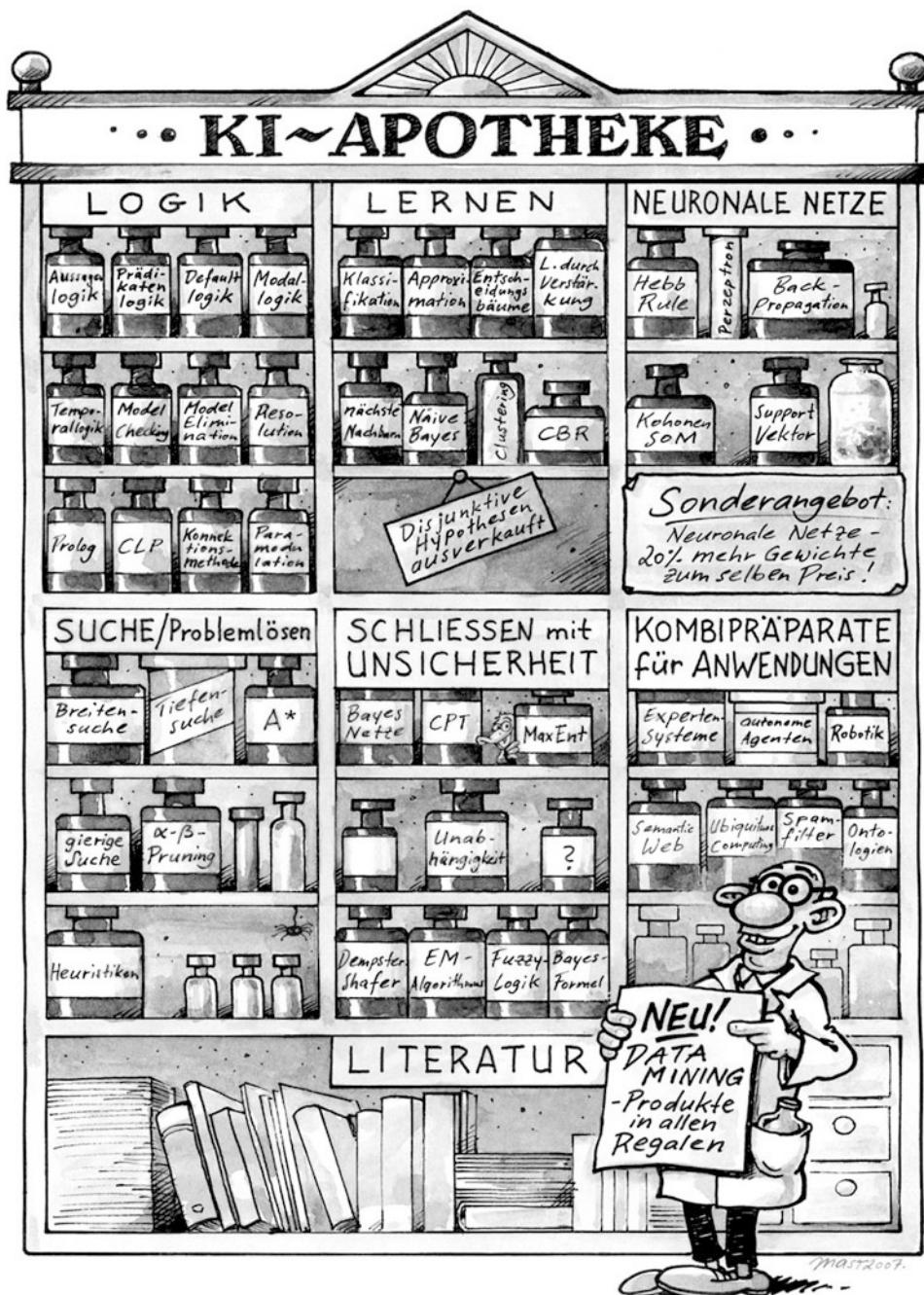


Abb. 1.2 Ein kleiner Ausschnitt aus dem Angebot an KI-Verfahren

Antworten beeindrucken. Nach einer gewissen Zeit wird jedoch ihre beschränkte künstliche Natur deutlich. Manche dieser Programme sind sogar lernfähig, andere besitzen ein erstaunliches Wissen über verschiedene Wissensbereiche, zum Beispiel Geographie oder Software-Entwicklung. Eine interessante zukünftige Anwendung für Chatterbots könnte das Gebiet des E-Learning sein. Denkbar wäre etwa eine Kommunikation des Lernenden mit dem E-Learning-System über solch einen Chatterbot. Der Leser möge sich in Aufgabe 1.1 mit einigen Chatterbots auseinandersetzen und selbst deren Intelligenz beurteilen.

1.2 Geschichte der KI

Die KI greift auf viele alte wissenschaftliche Errungenschaften zurück, die hier nicht erwähnt werden, denn die KI als eigene Wissenschaft gibt es erst seit Mitte des zwanzigsten Jahrhunderts. Der folgende Text wird ergänzt durch Tab. 1.1 mit den wichtigsten Meilensteinen der KI und eine grafische Darstellung der Hauptströmungen der KI in Abb. 1.3.

1.2.1 Die ersten Anfänge

In den dreißiger Jahren des zwanzigsten Jahrhunderts sind durch Kurt Gödel, Alonso Church und Alan Turing wichtige Fundamente für die Logik und die theoretische Informatik gelegt worden. Für die KI von besonderer Bedeutung sind die Gödelschen Sätze. Der Vollständigkeitssatz besagt, dass die Prädikatenlogik erster Stufe vollständig ist. Das heißt, jede in der Prädikatenlogik formalisierbare wahre Aussage ist beweisbar mit Hilfe der Schlussregeln eines formalen Kalküls. Auf diesem Fundament konnten später dann die automatischen Theorembeweiser als Implementierung formaler Kalküle gebaut werden. Mit dem Unvollständigkeitssatz zeigte Gödel, dass es in Logiken höherer Stufe wahre Aussagen gibt, die nicht beweisbar sind.² Er zeigte damit eine oft schmerzhafte Grenze formaler Systeme auf.

Auch in diese Zeit fällt Alan Turings Beweis der Unentscheidbarkeit des Halteproblems. Er zeigte, dass es kein Programm geben kann, das für beliebige Programme (und zugehörige Eingabe) entscheiden kann, ob dieses in eine Endlosschleife läuft. Auch Turing zeigt damit eine Grenze für intelligente Programme auf. Es folgt daraus zum Beispiel, dass es nie ein universelles Programmverifikationssystem geben wird.³

In den vierziger Jahren wurden dann, basierend auf Ergebnissen aus der Hirnforschung, durch McCulloch, Pitts und Hebb die ersten mathematischen Modelle für neuronale Netze

² Logiken höherer Stufe sind Erweiterungen der Prädikatenlogik, in denen nicht nur über Variablen, sondern auch über Funktionssymbole oder Prädikate quantifiziert werden kann. Gödel zeigte allerdings nur, dass jedes System, das auf der Prädikatenlogik basiert und die Peanoarithmetic formalisieren kann, unvollständig ist.

³ Diese Aussage gilt für die „totale Korrektheit“, die neben einem Beweis der korrekten Ausführung eben auch einen Beweis der Terminierung für jede erlaubte Eingabe beinhaltet.

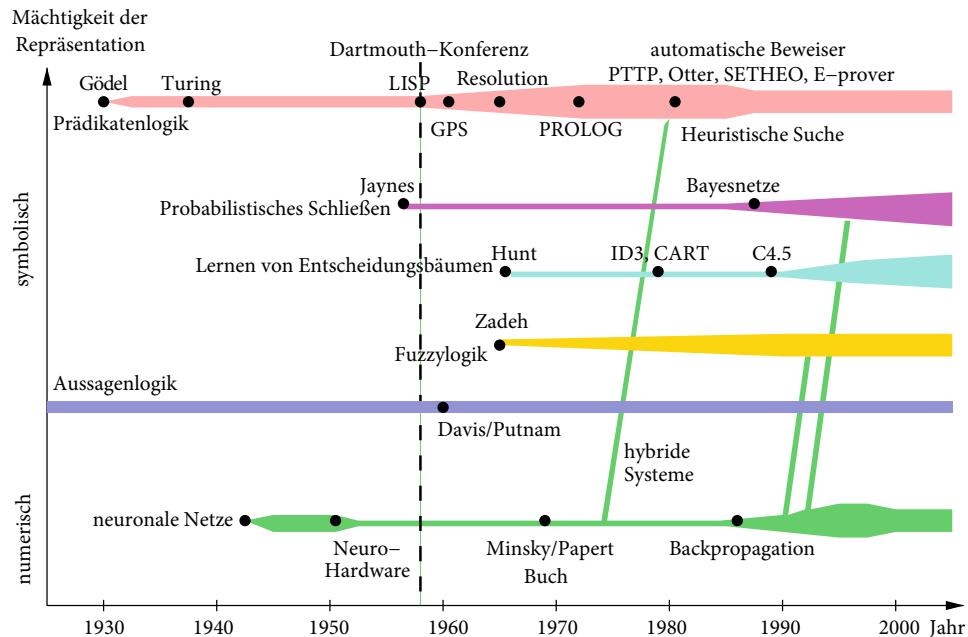


Abb. 1.3 Die Geschichte der verschiedenen KI-Ausrichtungen. Die Breite der Balken soll die Verbreitung der Methode in Anwendungen andeuten

entworfen. Zur Simulation einfacher Gehirne fehlten zu dieser Zeit aber noch leistungsfähige Computer.

1.2.2 Logik löst (fast) alle Probleme

Die KI als praktische Wissenschaft der Mechanisierung des Denkens konnte natürlich erst beginnen, als es programmierbare Rechenmaschinen gab. Dies war in den fünfziger Jahren der Fall. Newell und Simon stellten Logic Theorist, den ersten automatischen Theorembeweiser, vor und zeigten damit auch, dass man mit Computern, die eigentlich nur mit Zahlen arbeiten, auch Symbole verarbeiten kann. Gleichzeitig stellte McCarthy mit der Sprache LISP eine Programmiersprache vor, die speziell für die Verarbeitung von symbolischen Strukturen geschaffen wurde. Diese beiden Systeme wurden 1956 auf der historischen Dartmouth-Konferenz vorgestellt, welche als Geburtsstunde der KI gilt.

LISP entwickelte sich in den USA zum wichtigsten Werkzeug für die Implementierung von symbolverarbeitenden KI-Systemen. Es folgt die Entwicklung der Resolution als vollständigem Kalkül für die Prädikatenlogik.

In den siebziger Jahren wird die Logikprogrammiersprache Prolog als europäisches Pendant zu LISP vorgestellt. Prolog bietet den Vorteil, dass hier die Hornklauseln als Teil-

menge der Prädikatenlogik direkt zum Programmieren verwendet werden. Wie LISP besitzt auch Prolog Datentypen zur komfortablen Verarbeitung von Listen (Kap. 5).

Bis weit in die achtziger Jahre hinein herrschte in der KI, insbesondere bei vielen Logikern, Aufbruchstimmung. Grund hierfür waren eine Reihe von beeindruckenden Erfolgen der Symbolverarbeitung. Mit dem Fifth Generation Programme in Japan und dem ESPRIT-Programm in Europa wurde kräftig in den Bau intelligenter Computer investiert.

An kleinen Problemen funktionierten automatische Beweiser und andere symbolverarbeitende Systeme teilweise sehr gut. Die kombinatorische Explosion der Suchräume jedoch setzte den Erfolgen ganz enge Grenzen. Diese Phase der KI wurde in [RN03] als die „Look, Ma, no hands!“⁴-Ära bezeichnet.

Da der wirtschaftliche Erfolg symbolischer KI-Systeme aber hinter den Erwartungen zurückblieb, wurde während den achtziger Jahren in den USA die Förderung für logikbasierte KI-Forschung stark reduziert.

1.2.3 Der neue Konnektionismus

In dieser Phase der Ernüchterung konnten Informatiker, Physiker und Kognitionswissenschaftler mit Hilfe der nunmehr leistungsfähigen Computer zeigen, dass die mathematisch modellierten neuronalen Netze in der Lage sind, anhand von Trainingsbeispielen, Aufgaben zu erlernen, die man zuvor nur mit viel Aufwand programmieren konnte. Speziell in der Mustererkennung wurden durch die Fehlertoleranz der Netze und die Fähigkeit, Ähnlichkeiten zu erkennen, beachtliche Erfolge wie zum Beispiel das Erkennen von Personen anhand von Portraitfotos oder die Handschrifterkennung möglich (Kap. 9). Das System Nettalk konnte anhand von Beispieldaten das Sprechen lernen [SR86]. Unter dem Namen **Konnektionismus** war ein neues Teilgebiet der KI entstanden.

Der Konnektionismus boomte und die Fördermittel flossen. Aber bald wurden auch hier die Grenzen des Machbaren deutlich. Die neuronalen Netze konnten zwar eindrucksvolle Fähigkeiten lernen, aber es war meist nicht möglich, das gelernte Konzept in einfache Formeln oder logische Regeln zu fassen. Die Kombination von neuronalen Netzen mit logischen Regeln oder menschlichem Expertenwissen bereitete große Schwierigkeiten. Auch wurden keine befriedigenden Lösungen zur Strukturierung und Modularisierung der Netze gefunden.

1.2.4 Schließen mit Unsicherheit

Die KI als praktische, zielorientierte Wissenschaft suchte in dieser Krise nach Auswegen. Man wollte die Stärke der Logik, das Wissen explizit zu repräsentieren und die der neuronalen Netze, mit Unsicherheit umzugehen, vereinigen. Mehrere Auswege wurden vorgeschlagen.

⁴ Schau Mama, ich kann freihändig Rad fahren!

Der vielversprechendste, das **probabilistische Schließen**, arbeitet mit bedingten Wahrscheinlichkeiten für aussagenlogische Formeln. Mit Hilfe von **Bayes-Netzen** werden bis heute viele erfolgreiche Diagnose- und Expertensysteme für Probleme des Alltagsschließens gebaut (Abschn. 7.4). Der Erfolg der Bayes-Netze basiert auf der intuitiv leichten Verständlichkeit, der sauberen Semantik bedingter Wahrscheinlichkeiten und auf der Jahrhunderte alten, mathematisch fundierten Wahrscheinlichkeitstheorie.

Die Schwäche der Logik, nur mit zwei Wahrheitswerten umgehen zu können, löst die **Fuzzy-Logik** durch die pragmatische Einführung von unendlich vielen Werten zwischen null und eins. Auch wenn bis heute das theoretische Fundament nicht ganz fest ist [Elk93], wird sie, insbesondere in der Regelungstechnik, erfolgreich praktisch eingesetzt.

Ein ganz anderer Weg führte zur erfolgreichen Synthese von Logik und neuronalen Netzen unter dem Namen **Hybride Systeme**. Es wurden zum Beispiel neuronale Netze eingesetzt, um Heuristiken zur Reduktion der riesigen kombinatorischen Suchräume bei der Suche nach Beweisen zu lernen [SE90].

Auch mit Wahrscheinlichkeiten arbeiten die Methoden zum Lernen von Entscheidungsbäumen aus Daten. Systeme wie CART, ID3 oder C4.5 können sehr schnell und automatisch Entscheidungsbäume mit sehr guter Korrektheit aufbauen, die aussagenlogische Konzepte repräsentieren können und dann als Expertensysteme einsetzbar sind. Sie gehören heute zu den beliebtesten maschinellen Lernverfahren (Abschn. 8.4).

Im Umfeld der statistischen Datenanalyse zur Gewinnung von Wissen aus großen Datenbanken hat sich seit etwa 1990 das **Data Mining** als neue Teildisziplin der KI entwickelt. Data Mining bringt keine neuen Verfahren in die KI, sondern die Anforderung, aus großen Datenbanken explizites Wissen zu gewinnen. Eine Anwendung mit großem Marktpotenzial ist die Steuerung von Werbeaktionen großer Handelshäuser basierend auf der Analyse vieler Millionen von Einkäufen ihrer Kunden. Typischerweise kommen hier maschinelle Lernverfahren, zum Beispiel das Lernen von Entscheidungsbäumen, zum Einsatz.

1.2.5 Verteilte, autonome und lernende Agenten

Seit etwa 1985 werden verteilte KI-Systeme (engl. distributed artificial intelligence, DAI) erforscht. Eine Zielsetzung ist die Nutzung von Parallelrechnern zur Effizienzsteigerung von Problemlösern. Es hat sich jedoch gezeigt, dass aufgrund der hohen Berechnungskomplexität für die meisten Probleme die Verwendung „intelligenterer“ Verfahren mehr Gewinn bringt als die Parallelisierung.

Eine ganz andere Aufgabenstellung ergibt sich bei der Entwicklung autonomer Software-Agenten und Roboter, die wie menschliche Teams bei der Bearbeitung ihrer Aufgaben kooperieren sollen. Wie bei den erwähnten Braitenberg-Vehikeln gibt es viele Beispiele, bei denen der einzelne Agent nicht in der Lage ist, ein Problem zu lösen, auch nicht mit beliebigem Aufwand. Erst die Zusammenarbeit vieler Agenten führt zu dem intelligenten Verhalten oder zur Lösung eines Problems. Ein Ameisenvolk oder ein Termitenvolk etwa ist in der Lage, Gebäude von sehr hoher architektonischer Komplexität zu errichten, obwohl

Tab. 1.1 Meilensteine in der Entwicklung der KI von Gödel bis heute

1931	Der Österreicher Kurt Gödel zeigt, dass in der Prädikatenlogik erster Stufe alle wahren Aussagen herleitbar sind [Göd31a]. In Logiken höherer Stufe hingegen gibt es wahre Aussagen, die nicht beweisbar sind [Göd31b, Gue02]. ^a
1937	Alan Turing zeigt mit dem Halteproblem Grenzen intelligenter Maschinen auf [Tur37].
1943	McCulloch und Pitts modellieren neuronale Netze und stellen die Verbindung zur Aussagenlogik her.
1950	Alan Turing definiert über den Turing-Test Intelligenz von Maschinen und schreibt über lernende Maschinen und genetische Algorithmen [Tur50].
1951	Marvin Minsky entwickelt einen Neuronenrechner. Mit 3000 Röhren simuliert er 40 Neuronen.
1955	Arthur Samuel (IBM) baut lernfähige Dame Programme, die besser spielen als ihre Entwickler [Sam59].
1956	McCarthy organisiert eine Konferenz im Dartmouth College. Hier wird der Name Artificial Intelligence eingeführt. Newell und Simon von der Carnegie Mellon University (CMU) stellen den <i>Logic Theorist</i> , das erste symbolverarbeitende Programm, vor [NSS83].
1958	McCarthy erfindet am MIT (Massachusetts Institute of Technology) die Hochsprache LISP . Er schreibt Programme, die sich selbst verändern können.
1959	Gelernter (IBM) baut 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 Universität.
1965	Robinson erfindet den Resolutionskalkül für Prädikatenlogik [Rob65] (Abschn. 3.5).
1966	Weizenbaum's Eliza-Programm führt Dialoge mit Menschen in natürlicher Sprache [Wei66] (Abschn. 1.1.2).
1969	Minsky und Papert zeigen in ihrem Buch <i>Perceptrons</i> auf, dass das Perzeptron, ein sehr einfaches neuronales Netz, nur lineare Zusammenhänge repräsentieren kann [MP69] (Abschn. 1.1.2).
1972	Der Franzose Alain Colmerauer erfindet die Logikprogrammiersprache Prolog (Kap. 5). Der britische Mediziner de Dombal entwickelt ein Expertensystem zur Diagnose von Bauchkrankheiten [dLDS ⁺ 72]. Es blieb in der bis dahin überwiegend amerikanischen KI-Community unbeachtet (Abschn. 7.3).
1976	Shortliffe und Buchanan entwickeln MYCIN, ein Expertensystem zur Diagnose von Infektionskrankheiten, das mit Unsicherheit umgehen kann (Kap. 7).
1981	Japan startet mit großem Aufwand das „Fifth Generation Project“ mit dem Ziel, leistungsfähige intelligente Prolog-Maschinen zu bauen.
1982	Das Expertensystem R1 zur Konfiguration von Computern spart der Digital Equipment Corporation 40 Millionen Dollar pro Jahr [McD82].
1986	Renaissance der neuronalen Netze unter anderem durch Rumelhart, Hinton und Sejnowski [RM86]. Das System Nettalk lernt das Vorlesen von Texten. [SR86] (Kap. 9).

^a In [Göd31b] hat Gödel gezeigt, dass die Prädikatenlogik erweitert um die Axiome der Arithmetik unvollständig ist.

Tab. 1.1 (Fortsetzung)

1990	Pearl [Pea88], Cheeseman [Che85], Whittaker, Spiegelhalter bringen mit den Bayes-Netzen die Wahrscheinlichkeitstheorie in die KI (Abschn. 7.4).
	Multiagentensysteme werden populär.
1992	Tesauros TD-Gammon Programm zeigt die Stärke des Lernens durch Verstärkung auf.
1993	Weltweite RoboCup Initiative zum Bau Fußball spielender autonomer Roboter [Roba].
1995	Vapnik entwickelt aus der statistischen Lerntheorie die heute wichtigen Support-Vektor-Maschinen.
1997	Erster internationaler RoboCup Wettkampf in Japan.
2003	Die Roboter im RoboCup demonstrieren eindrucksvoll, was KI und Robotik zu leisten imstande sind.
2006	Servicerobotik entwickelt sich zu einem dominanten Forschungsgebiet in der KI.
2010	Autonome Roboter fangen an, ihr Verhalten zu lernen.
2011	Die IBM-Software „Watson“ schlägt zwei menschliche Meister in der US-Fernsehshow „Jeopardy!“. Watson kann natürliche Sprache verstehen und sehr schnell Fragen beantworten (Abschn. 1.4).

keine einzige Ameise ein Verständnis der globalen Zusammenhänge hat. Ähnlich ist zum Beispiel die Situation bei der Versorgung einer Großstadt wie New York mit Brot [RN03]. Es gibt keinen zentralen Brotbedarfsplaner, sondern Hunderte von Bäckern, die ihren jeweiligen Stadtbezirk kennen und dort die passende Menge an Brot backen.

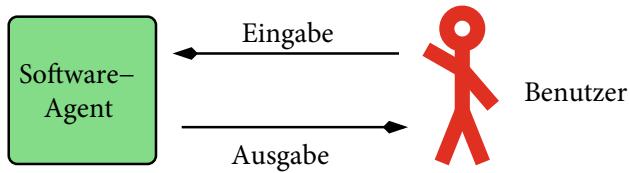
Ein spannendes aktuelles Forschungsgebiet ist das aktive Lernen von Fähigkeiten durch Roboter. Zum Beispiel gibt es heute Roboter, die selbständig das Laufen lernen oder verschiedene motorische Fähigkeiten beim Fußballspielen lernen (Kap. 10). Noch in den Anfängen ist das kooperative Lernen mehrerer Roboter, die gemeinsam eine Aufgabe lösen sollen.

1.2.6 Die KI wird erwachsen

Heute bietet die KI mit den erwähnten Verfahren zwar kein Universalrezept, aber eine Werkstatt mit einer überschaubaren Anzahl an Werkzeugen für die unterschiedlichsten Aufgaben. Die meisten dieser Werkzeuge sind mittlerweile weit entwickelt und als fertige Software-Bibliotheken, oft mit komfortabler Benutzeroberfläche, verfügbar. Die Auswahl des richtigen Werkzeugs und seine sinnvolle Anwendung im Einzelfall obliegt dem KI-Entwickler, beziehungsweise dem Wissens-Ingenieur (engl. knowledge engineer). Wie in jedem anderen Handwerk bedarf es auch hier einer soliden Ausbildung, zu der dieses Buch beitragen soll.

Die KI ist wie kaum eine andere Wissenschaft interdisziplinär, denn sie nutzt viele interessante Ergebnisse aus so unterschiedlichen Gebieten wie Logik, Operations Research, Statistik, Regelungstechnik, Bildverarbeitung, Linguistik, Philosophie, Psychologie und Neu-

Abb. 1.4 Ein Software-Agent mit Benutzerinteraktion



robiologie. Hinzu kommt in Anwendungsprojekten noch das Fachgebiet der jeweiligen Anwendung. KI-Projekte erfolgreich zu bearbeiten ist daher nicht immer ganz einfach, aber fast immer äußerst spannend.

1.3 Agenten

Obwohl der Begriff des *intelligenten Agenten* in der KI schon alt ist, hat er sich erst in den letzten Jahren unter anderem durch [RN03] richtig durchgesetzt. Als **Agent** bezeichnen wir ganz allgemein ein System, welches Information verarbeitet und aus einer Eingabe eine Ausgabe produziert. Diese Agenten lassen sich in vielfältiger Weise klassifizieren.

In der klassischen Informatik werden hauptsächlich **Software-Agenten** verwendet (Abb. 1.4). Der Agent besteht hier aus einem Programm, das aus Benutzereingaben ein Ergebnis berechnet.

In der Robotik hingegen werden **Hardware-Agenten** (auch Roboter genannt) verwendet, die zusätzlich über Sensoren und Aktuatoren verfügen (Abb. 1.5). Mit den Sensoren kann der Agent die Umgebung wahrnehmen. Mit den Aktuatoren führt er Aktionen aus und verändert so die Umgebung.

Bezüglich der Intelligenz des Agenten unterscheidet man zwischen einfachen **Reflex-Agenten**, die nur auf die Eingabe reagieren, und **Agenten mit Gedächtnis**, die bei ihren Entscheidungen auch die Vergangenheit mit einbeziehen können. Zum Beispiel hat ein fahrender Roboter, der über die Sensoren seinen genauen Ort (und die Zeit) kennt, als Reflex-Agent keine Chance, seine Geschwindigkeit zu bestimmen. Speichert er jedoch den Ort in kurzen diskreten Zeitabständen jeweils bis zum nächsten Zeitpunkt, so kann er ganz einfach seine mittlere Geschwindigkeit im letzten Zeitintervall berechnen.

Wird ein Reflex-Agent durch ein deterministisches Programm gesteuert, so repräsentiert er eine Funktion von der Menge aller Eingaben auf die Menge aller Ausgaben. Ein Agent mit Gedächtnis hingegen ist im Allgemeinen keine Funktion. Warum? (siehe Aufgabe 1.5). Reflex-Agenten sind ausreichend, falls es sich bei der zu lösenden Aufgabe um einen Markov-Entscheidungsprozess handelt. Dies ist ein Prozess, bei dem zur Bestimmung der optimalen Aktion nur der aktuelle Zustand benötigt wird (siehe Kap. 10).

Ein mobiler Roboter, der sich in einem Gebäude von Raum 112 nach Raum 179 bewegen soll, wird andere Aktionen ausführen als einer, der sich nach Raum 105 bewegen soll. Die Aktionen hängen also vom Ziel ab. Solche Agenten heißen **zielorientiert**.

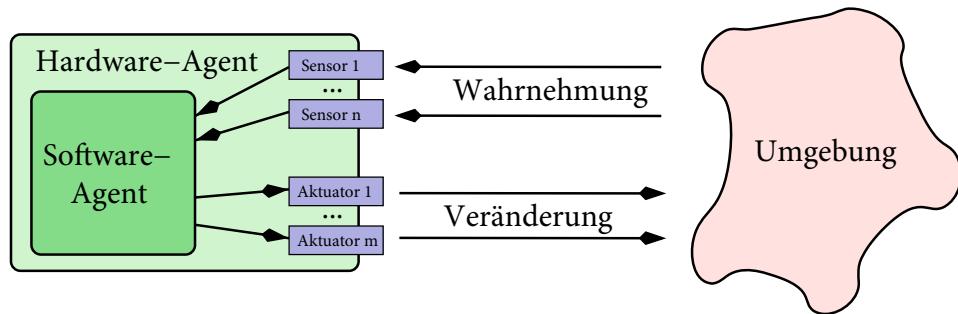


Abb. 1.5 Ein Hardware-Agent

Beispiel 1.1

Ein Spam-Filter ist ein Agent, der ankommende Emails in erwünschte und unerwünschte (Spam) einteilt und die unerwünschten Emails gegebenenfalls löscht. Sein Ziel als zielorientierter Agent ist es, alle Emails in die richtige Klasse einzuteilen. Bei dieser nicht ganz einfachen Aufgabe kann es vorkommen, dass der Agent ab und zu Fehler macht. Da es sein Ziel ist, alle Emails korrekt zu klassifizieren, wird er versuchen, möglichst wenige Fehler zu machen. Das ist jedoch nicht immer im Sinne der Benutzers. Vergleichen wir folgende zwei Agenten. Bei 1000 Emails macht Agent 1 nur 12 Fehler. Agent 2 hingegen macht bei diesen 1000 Emails 38 Fehler. Ist er damit schlechter als Agent 1? Die Fehler der beiden Agenten sind in folgenden Tabellen, den so genannten Wahrheitstafeln (engl. confusion matrix), genauer dargestellt:

Agent 1:

		korrekte Klasse	
		erwünscht	Spam
Spam-Filter entscheidet	erwünscht	189	1
	Spam	11	799

Agent 2:

		korrekte Klasse	
		erwünscht	Spam
Spam-Filter entscheidet	erwünscht	200	38
	Spam	0	762

Agent 1 macht zwar weniger Fehler als Agent 2, aber die wenigen Fehler sind schwerwiegend, denn dem Benutzer gehen 11 eventuell wichtige Emails verloren. Da es hier zwei unterschiedlich schwerwiegende Arten von Fehlern gibt, sollten diese Fehler mit einem Kostenfaktor entsprechend gewichtet werden (siehe Abschn. 7.3.5 und Aufgabe 1.7).

Die Summe aller gewichteten Fehler ergibt dann die durch die Fehlentscheidungen entstandenen Kosten. Ziel eines **kostenorientierten Agenten** ist es also, die durch Fehlentscheidungen entstehenden Kosten langfristig, das heißt im Mittel, zu minimieren. In Abschn. 7.3 werden wir am Beispiel des Blinddarmdiagnosesystems LEXMED einen kostenorientierten Agenten kennenlernen.

Analog ist das Ziel eines **nutzenorientierten Agenten** (engl. utility based agent), den durch korrekte Entscheidungen entstehenden Nutzen langfristig, das heißt im Mittel, zu maximieren. Die Summe aller mit ihrem jeweiligen Nutzenfaktor gewichteten Entscheidungen ergibt dann den Nutzen.

Für die KI von besonderem Interesse sind **lernfähige Agenten**, die anhand von Trainingsbeispielen erfolgreicher Aktionen oder durch positives oder negatives Feedback auf die Aktionen in der Lage sind, sich selbst so zu verändern, dass der mittlere Nutzen ihrer Aktionen im Laufe der Zeit wächst (siehe Kap. 8).

Wie in Abschn. 1.2.5 schon erwähnt, kommen heute vermehrt auch **verteilte Agenten** zum Einsatz, bei denen die Intelligenz nicht in einem Agenten lokalisiert ist, sondern erst durch die Kooperation vieler Agenten in Erscheinung tritt.

Der Entwurf eines Agenten orientiert sich neben der Aufgabenstellung stark an seiner **Umgebung** beziehungsweise seinem Bild der Umgebung, das stark von den Sensoren abhängt. Man nennt eine Umgebung **beobachtbar**, wenn der Agent immer den kompletten Zustand der Welt kennt. Andernfalls ist die Umgebung nur **teilweise beobachtbar**. Führt eine Aktion immer zum gleichen Ergebnis, so ist die Umgebung **deterministisch**. Andernfalls ist sie **nichtdeterministisch**. In einer **diskreten Umgebung** kommen nur endlich viele Zustände und Aktionen vor, wogegen eine **stetige Umgebung** unendlich viele Zustände oder Aktionen aufweist.

1.4 Wissensbasierte Systeme

Ein Agent ist ein Programm, das seine Wahrnehmungen auf Aktionen abbildet. Für sehr einfache Agenten ist diese Sichtweise ausreichend. Für komplexe Anwendungen, bei denen der Agent auf eine große Menge von Wissen zurückgreifen muss und schwierige Aufgabenstellungen hat, kann das Programmieren des Agenten sehr aufwändig und unübersichtlich werden. In der KI geht man hier einen strukturierten Weg, der die Arbeit stark vereinfacht.

Zuerst trennt man das **Wissen** von dem Verfahren oder Programm, welches das Wissen verwendet, um zum Beispiel Schlüsse zu ziehen, Anfragen zu beantworten oder einen Plan zu erstellen. Dieses Verfahren nennt man **Inferenzmechanismus**. Das Wissen wird in einer **Wissensbasis**, kurz **WB**, (engl. knowledge base, KB) gespeichert. Der Erwerb des Wissens in der Wissensbasis wird als **Knowledge Engineering** bezeichnet und basiert auf unterschiedlichen Wissensquellen wie zum Beispiel menschlichen Experten, dem Wissenschaftsingenieur (engl. knowledge engineer) und auch auf Datenbanken. Aktiv lernende Systeme können auch durch aktive Exploration der Welt Wissen erwerben (siehe Kap. 10). In Abb. 1.6 ist die allgemeine Architektur wissensbasierter Systeme dargestellt.

Das Vorgehen der Trennung von Wissen und Inferenz hat mehrere entscheidende Vorteile. Durch die Trennung von Wissen und Inferenz können Inferenzverfahren weitgehend anwendungsunabhängig implementiert werden. Zum Beispiel ist es bei einem medizinischen Expertensystem viel einfacher, die Wissensbasis auszutauschen, um es auch auf andere Krankheiten anzuwenden, als das ganze System neu zu programmieren.

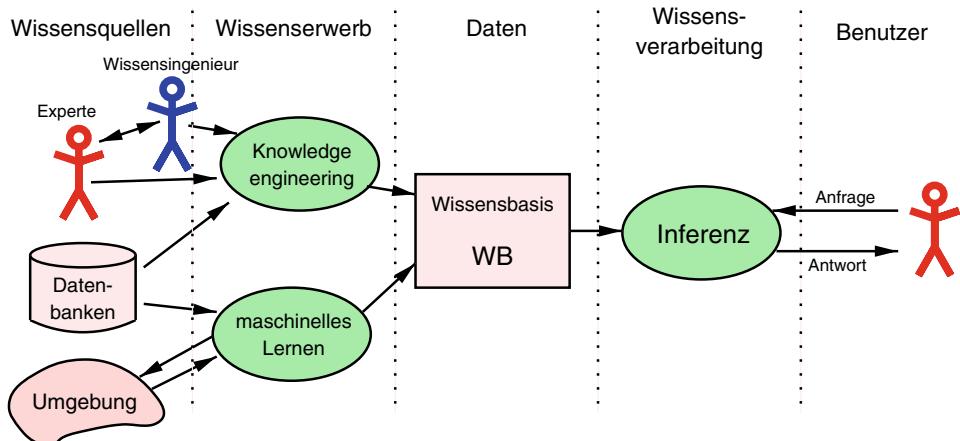


Abb. 1.6 Struktur eines klassischen wissensverarbeitenden Systems

Durch das Abkoppeln der Wissensbasis von der Inferenz kann das Wissen deklarativ gespeichert werden. In der Wissensbasis steht nur eine Beschreibung des Wissens, die unabhängig vom verwendeten Inferenzverfahren ist. Ohne diese klare Trennung wären Wissen und Abarbeitung von Inferenzschritten verwoben und die Änderung des Wissens wäre sehr aufwändig.

Zur Darstellung des Wissens in der Wissensbasis als Schnittstelle zwischen Mensch und Maschine bietet sich die Verwendung einer formalen Sprache an. Wir werden in den folgenden Kapiteln eine ganze Reihe von derartigen Sprachen kennenlernen. An erster Stelle stehen in den Kap. 2 und 3 die **Aussagenlogik** und die **Prädikatenlogik erster Stufe** (kurz: PL1). Aber auch andere Formalismen wie probabilistische Logik, Fuzzy-Logik oder Entscheidungsbäume werden vorgestellt. Wir starten mit der Aussagenlogik und den zugehörigen Inferenzverfahren. Darauf aufbauend wird mit der Prädikatenlogik eine mächtige, der maschinellen Verarbeitung noch zugängliche, und für die KI sehr wichtige Sprache behandelt.

Der von IBM zusammen mit mehreren Universitäten entwickelte Software-Agent „Watson“ stellt ein beeindruckendes Beispiel für ein großes und erfolgreiches wissensbasiertes System dar. Watson ist ein Programm, das Fragen in gesprochener natürlicher Sprache beantworten kann. Es nutzt eine vier Terabyte große Wissensbasis, die unter anderem den kompletten Volltext von Wikipedia beinhaltet [FNA⁺09]. In [Dee11] wird Watson wie folgt beschrieben:

The DeepQA project at IBM shapes a grand challenge in Computer Science that aims to illustrate how the wide and growing accessibility of natural language content and the integration and advancement of Natural Language Processing, Information Retrieval, Machine Learning, Knowledge Representation and Reasoning, and massively parallel computation can drive open-domain automatic Question Answering technology to a point where it clearly and consistently rivals the best human performance.

In der US-Fernsehshow „Jeopardy!“, im Februar 2011 besiegte Watson die zwei menschlichen Meister Brad Rutter und Ken Jennings nach zwei Spielen und gewann den Preis von einer Million Dollar. Eine von Watson's speziellen Stärken war seine schnelle Reaktion auf Fragen mit dem Ergebnis, dass Watson oft den Knopf für die Hupe (mittels eines Elektromagneten) schneller betätigte als seine menschlichen Gegner und damit die erste Antwort auf die Frage geben konnte.

Seine hohe Leistungsfähigkeit und seine kurze Reaktionszeit verdankte Watson auch einer Implementierung auf 90 IBM Power 750 Servern, jeder mit 32 Prozessoren, insgesamt also 2880 parallel arbeitenden Rechnern.

1.5 Übungen

Aufgabe 1.1 Testen Sie einige der im Netz verfügbaren Chatterbots auf deren Intelligenz. Starten Sie zum Beispiel auf <http://www.hs-weingarten.de/~ertel/kibuch> in der Linkssammlung unter Turingtest/Chatterbots oder auf <http://www.simonlaven.com>, bzw. <http://www.alicebot.org>. Notieren Sie sich eine Startfrage und messen Sie die Zeit, die Sie bei den verschiedenen Programmen benötigen, bis Sie sicher sagen können, dass es kein Mensch ist.

Aufgabe 1.2 *** Unter <http://www.pandorabots.com> finden Sie einen Server, auf dem Sie mit der Markup-Sprache AIML selbst recht einfach einen Chatterbot bauen können. Entwerfen Sie, je nach Interesse, einen einfachen oder auch komplexeren Chatterbot oder verändern Sie einen bestehenden.

Aufgabe 1.3 Geben Sie Gründe an für die Untauglichkeit des Turing-Tests zur Definition von „*Künstliche Intelligenz*“ in der praktischen KI.

Aufgabe 1.4 => Viele bekannte Inferenzverfahren, Lernverfahren, etc. sind NP-vollständig oder sogar unentscheidbar. Was bedeutet das für die KI?

Aufgabe 1.5

- Warum ist ein deterministischer Agent mit Gedächtnis keine Funktion von der Menge aller Eingaben auf die Menge aller Ausgaben im mathematischen Sinn?
- Wie kann man den Agenten mit Gedächtnis verändern, beziehungsweise modellieren, so dass er funktional wird, aber sein Gedächtnis nicht verliert.

Aufgabe 1.6 Gegeben sei ein Agent mit Gedächtnis, der sich in der Ebene bewegen kann. Er verfügt über Sensoren, die ihm zeitlich getaktet immer im Abstand Δt die exakte Position (x, y) in kartesischen Koordinaten liefern.

- Geben Sie eine Formel an, mit der der Agent aus der aktuellen Messung zum Zeitpunkt t und der vorhergehenden Messung bei $t - \Delta t$ seine Geschwindigkeit schätzen kann.
- Wie muss der Agent verändert werden, so dass er auch noch die Beschleunigung schätzen kann? Geben Sie auch hier eine Formel an.

Aufgabe 1.7 *

- a) Bestimmen Sie für die beiden Agenten aus Beispiel 1.1 die durch die Fehlentscheidungen entstandenen Kosten und vergleichen Sie die Ergebnisse. Nehmen Sie hierzu an, für das Löschen einer Spam-Mail fallen Kosten in Höhe von 1 Cent und für das Wiederbeschaffen gelöschter Mails, bzw. den Verlust einer Mail fallen Kosten in Höhe von 1 Euro an.
- b) Bestimmen Sie nun für die beiden Agenten den durch korrekte Klassifikationen entstandenen Nutzen und vergleichen Sie die Ergebnisse. Nehmen Sie an, für jede korrekt erkannte erwünschte Email entsteht ein Nutzen von 1 Euro und für jede korrekt gelöschte Spam-Mail ein Nutzen von 1 Cent.

In der Aussagenlogik werden, wie der Name schon sagt, Aussagen über logische Operatoren verknüpft. Der Satz „*die Straße ist nass*“ ist eine Aussage, genauso wie „*es regnet*“. Diese beiden Aussagen lassen sich nun verknüpfen zu der neuen Aussage

wenn es regnet ist die Straße nass.

Etwas formaler geschrieben ergibt sich

es regnet \Rightarrow die Straße ist nass.

Diese Schreibweise hat unter anderem den Vorteil, dass die elementaren Aussagen in unveränderter Form wieder auftreten. Um im Folgenden ganz exakt mit der Aussagenlogik arbeiten zu können, starten wir mit einer Definition der Menge aller aussagenlogischen Formeln.

2.1 Syntax

Definition 2.1

Sei $Op = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, (,)\}$ die Menge der logischen Operatoren und Σ eine Menge von Symbolen. Die Mengen Op , Σ und $\{w, f\}$ seien paarweise disjunkt. Σ heißt **Signatur** und seine Elemente sind die **Aussagevariablen**. Die Menge der Aussagenlogischen Formeln wird nun rekursiv definiert:

- w und f sind (atomare) Formeln.
- Alle Aussagevariablen, das heißt alle Elemente von Σ sind (atomare) Formeln.
- Sind A und B Formeln, so sind auch $\neg A$, (A) , $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ Formeln.

Definition 2.2

Man liest die Operatoren und Symbole folgendermaßen:

w :	„wahr“
f :	„falsch“
$\neg A$:	„nicht A “ (Negation)
$A \wedge B$:	„ A und B “ (Konjunktion)
$A \vee B$:	„ A oder B “ (Disjunktion)
$A \Rightarrow B$:	„wenn A dann B “ (Implikation ¹)
$A \Leftrightarrow B$:	„ A genau dann, wenn B “ (Äquivalenz)

Diese elegante rekursive Definition der Menge aller Formeln erlaubt uns nun die Erzeugung von unendlich vielen Formeln. Mit $\Sigma = \{A, B, C\}$ sind zum Beispiel

$$A \wedge B, \quad A \wedge B \wedge C, \quad A \wedge A \wedge A, \quad C \wedge B \vee A, \quad (\neg A \wedge B) \Rightarrow (\neg C \vee A)$$

Formeln. Auch $((A)) \vee B$ ist eine syntaktisch korrekte Formel.

Die so definierten Formeln sind bisher rein syntaktische Gebilde ohne Bedeutung. Es fehlt noch die Semantik.

2.2 Semantik

In der Aussagenlogik gibt es die zwei Wahrheitswerte w für „wahr“ und f für „falsch“. Starten wir mit einem Beispiel und fragen uns, ob die Formel $A \wedge B$ wahr ist. Die Antwort heißt: Es kommt darauf an, ob die Variablen A und B wahr sind. Steht zum Beispiel A für „Es regnet heute.“ und B für „Es ist kalt heute.“ und dies trifft beides zu, so ist $A \wedge B$ wahr. Steht jedoch B für „Es ist heiß heute.“ (und dies trifft nicht zu) so ist $A \wedge B$ falsch.

Wir müssen offenbar den Aussagevariablen Wahrheitswerte zuordnen, die Zuständen der Welt entsprechen. Daher definieren wir

Definition 2.3

Eine Abbildung $I : \Sigma \rightarrow \{w, f\}$, die jeder Aussagevariablen einen Wahrheitswert zuordnet, heißt **Belegung** oder **Interpretation** oder auch **Welt**.

Da jede Aussagevariable zwei Wahrheitswerte annehmen kann, besitzt eine aussagenlogische Formel mit n verschiedenen Variablen 2^n verschiedene Belegungen, beziehungs-

¹ Auch **materiale Implikation** genannt.

Tab. 2.1 Definition der logischen Operatoren per Wahrheitstabelle

A	B	(A)	$\neg A$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
w	w	w	f	w	w	w	w
w	f	w	f	f	w	f	f
f	w	f	w	f	w	w	f
f	f	f	w	f	f	w	w

weise Welten. Für die erlaubten Basisoperatoren definieren wir nun deren Wahrheitswerte, indem wir in der **Wahrheitstabelle** (siehe Tab. 2.1) für alle möglichen Belegungen einen Wahrheitswert angeben.

Die leere Formel ist unter allen Belegungen wahr. Um die Wahrheitswerte für komplexe Formeln zu bestimmen, müssen wir noch die Rangfolge der logischen Operatoren angeben. Falls Ausdrücke geklammert sind, ist immer zuerst der Term in der Klammer auszuwerten. Bei ungeklammerten Formeln sind die Prioritäten wie folgt geordnet, beginnend mit der stärksten Bindung: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.

Um die Äquivalenz von Formeln klar von der syntaktischen Äquivalenz zu unterscheiden, definieren wir:

Definition 2.4

Zwei Formeln F und G heißen semantisch äquivalent, wenn sie für alle Belegungen die gleichen Wahrheitswerte annehmen. Man schreibt $F \equiv G$.

Die semantische Äquivalenz wird vor allem dazu dienen, in der Metasprache, das heißt der natürlichen Sprache, über die Objektsprache, nämlich die Logik, zu reden. Die Aussage „ $A \equiv B$ “ etwa besagt, dass die beiden Formeln A und B semantisch äquivalent sind. Die Aussage „ $A \Leftrightarrow B$ “ hingegen ist ein syntaktisches Objekt der formalen Sprache der Aussagenlogik.

Je nachdem, in wievielen Welten eine Formel wahr ist, teilt man die Formeln ein in folgende Klassen.

Definition 2.5

Eine Formel heißt

- **erfüllbar**, falls sie bei mindestens einer Belegung wahr ist.
- **allgemeingültig** oder **wahr**, falls sie bei allen Belegungen wahr ist. Wahre Formeln heißen auch **Tautologien**.
- **unerfüllbar**, falls sie bei keiner Belegung wahr ist.

Jede erfüllende Belegung einer Formel heißt **Modell** der Formel.

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

Wir können nun auch für komplexere Formeln Wahrheitstafeln erstellen, um deren Wahrheitswerte zu ermitteln. Dies praktizieren wir sofort anhand einiger für die Praxis wichtiger Äquivalenzen von Formeln.

Satz 2.1

Die Operationen \wedge , \vee sind kommutativ und assoziativ und folgende Äquivalenzen sind allgemeingültig:

$$\begin{aligned}
 \neg A \vee B &\Leftrightarrow A \Rightarrow B && \text{(Implikation)} \\
 A \Rightarrow B &\Leftrightarrow \neg B \Rightarrow \neg A && \text{(Kontraposition)} \\
 (A \Rightarrow B) \wedge (B \Rightarrow A) &\Leftrightarrow (A \Leftrightarrow B) && \text{(Äquivalenz)} \\
 \neg(A \wedge B) &\Leftrightarrow \neg A \vee \neg B && \text{(De Morgan'sche Regeln)} \\
 \neg(A \vee B) &\Leftrightarrow \neg A \wedge \neg B \\
 A \vee (B \wedge C) &\Leftrightarrow (A \vee B) \wedge (A \vee C) && \text{(Distributivgesetze)} \\
 A \wedge (B \vee C) &\Leftrightarrow (A \wedge B) \vee (A \wedge C) \\
 A \vee \neg A &\Leftrightarrow w && \text{(Tautologie)} \\
 A \wedge \neg A &\Leftrightarrow f && \text{(Widerspruch)} \\
 A \vee f &\Leftrightarrow A \\
 A \vee w &\Leftrightarrow w \\
 A \wedge f &\Leftrightarrow f \\
 A \wedge w &\Leftrightarrow A
 \end{aligned}$$

Beweis Um die erste Äquivalenz zu zeigen, berechnen wir per Wahrheitstabelle $\neg A \vee B$ und $A \Rightarrow B$ und erkennen, dass die Wahrheitswerte beider Formeln in allen Welten (Belegungen) gleich sind. Die Formeln sind also äquivalent, und in der letzten Spalte stehen daher nur „w“-s.

A	B	$\neg A$	$\neg A \vee B$	$A \Rightarrow B$	$(\neg A \vee B) \Leftrightarrow (A \Rightarrow B)$
w	w	f	w	w	w
w	f	f	f	f	w
f	w	w	w	w	w
f	f	w	w	w	w

Die Beweise für die anderen Äquivalenzen laufen analog und werden dem Leser zur Übung empfohlen (Aufgabe 2.2). \square

2.3 Beweisverfahren

In der KI sind wir daran interessiert, aus bestehendem Wissen neues Wissen herzuleiten, beziehungsweise Anfragen zu beantworten. In der Aussagenlogik geht es darum zu zeigen, dass aus einer **Wissensbasis** WB , das heißt einer (eventuell umfangreichen) aussagenlogischen Formel, eine Formel Q^2 folgt. Zuerst definieren wir den Begriff der Folgerung.

Definition 2.6

Eine Formel Q **folgt** aus einer Formel WB , wenn jedes Modell von WB auch ein Modell von Q ist. Man schreibt dann $WB \vDash Q$.

In anderen Worten heißt das, dass in jeder Welt (Belegung von Variablen), in der WB wahr ist, auch Q wahr ist. Oder noch knapper: immer dann, wenn WB wahr ist, ist auch Q wahr. Da für den Folgerungsbegriff Belegungen von Variablen herangezogen werden, handelt es sich hier um einen semantischen Begriff.

Jede Formel, die nicht allgemeingültig ist, wählt gewissermaßen aus der Menge aller Belegungen eine Teilmenge als ihre Modelle aus. Tautologien wie zum Beispiel $A \vee \neg A$ schränken die Zahl der erfüllenden Belegungen nicht ein, denn ihre Aussage ist leer. Die leere Formel ist daher in allen Belegungen wahr. Für jede Tautologie T gilt also $\emptyset \vDash T$. Intuitiv heißt das, dass Tautologien immer gelten, ohne Einschränkung der Belegungen durch eine Formel. Man schreibt dann auch kurz $\vDash T$. Nun zeigen wir einen wichtigen Zusammenhang zwischen dem Folgerungsbegriff und der Implikation.

Satz 2.2 (Deduktionstheorem)

$$A \vDash B \text{ gilt genau dann wenn } \vDash A \Rightarrow B.$$

Beweis Betrachten wir die Wahrheitstabelle der Implikation:

A	B	$A \Rightarrow B$
w	w	w
w	f	f
f	w	w
f	f	w

² Q steht hier für Query (Anfrage).

Eine beliebige Implikation $A \Rightarrow B$ ist offenbar immer wahr, außer bei der Belegung $A \mapsto w, B \mapsto f$. Nehmen wir an, dass $A \vDash B$ gilt. Das heißt, dass für jede Belegung, die A wahr macht, auch B wahr ist. Die kritische zweite Zeile der Wahrheitstabelle kommt damit gar nicht vor. Also ist $A \Rightarrow B$ wahr, das heißt $A \Rightarrow B$ ist eine Tautologie. Damit ist eine Richtung des Satzes gezeigt.

Nehmen wir nun an, dass $A \Rightarrow B$ gilt. Damit ist die kritische zweite Zeile der Wahrheitstabelle auch ausgeschlossen. Jedes Modell von A ist dann auch Modell von B . Also gilt $A \vDash B$. \square

Will man zeigen, dass Q aus WB folgt, so kann man also mit Hilfe der Wahrheitstafelmethode nachweisen, dass $WB \Rightarrow Q$ eine Tautologie ist. Damit haben wir ein erstes **Beweisverfahren** für die Aussagenlogik, das sich leicht automatisieren lässt. Der Nachteil dieser Methode ist die im Worst-Case sehr lange Rechenzeit. Es muss nämlich im Worst-Case bei n Aussagevariablen für alle 2^n Belegungen der Variablen die Formel $WB \Rightarrow Q$ ausgewertet werden. Die Rechenzeit wächst also exponentiell mit der Zahl der Variablen. Somit ist dieses Verfahren für große Variablenanzahl zumindest im Worst-Case nicht anwendbar.

Wenn eine Formel Q aus WB folgt, so ist nach dem Deduktionstheorem $WB \Rightarrow Q$ eine Tautologie. Also ist die Negation $\neg(WB \Rightarrow Q)$ unerfüllbar. Wegen

$$\neg(WB \Rightarrow Q) \equiv \neg(\neg WB \vee Q) \equiv WB \wedge \neg Q$$

ist damit auch $WB \wedge \neg Q$ unerfüllbar. Diese einfache, aber wichtige Folgerung aus dem Deduktionstheorem formulieren wir als Satz.

Satz 2.3 (Widerspruchsbeweis)

$WB \vDash Q$ gilt genau dann wenn $WB \wedge \neg Q$ unerfüllbar ist.

Um zu zeigen, dass die zu beweisende Anfrage Q aus der Wissensbasis WB folgt, kann man also die negierte Anfrage $\neg Q$ zur Wissensbasis hinzufügen und daraus einen Widerspruch ableiten. Wegen der Äquivalenz $A \wedge \neg A \equiv f$ aus Satz 2.1 wissen wir, dass ein Widerspruch unerfüllbar ist. Damit ist also Q bewiesen. Dieses in der Mathematik häufig angewendete Verfahren wird auch von verschiedenen automatischen Beweiskalkülen, unter anderem vom Resolutionskalkül oder bei der Abarbeitung von Prolog-Programmen verwendet.

Eine Möglichkeit, das Durchprobieren aller Belegungen bei der Wahrheitstafelmethode zu vermeiden, ist die syntaktische Manipulation der Formeln WB und Q durch Anwendung von Inferenzregeln mit dem Ziel, diese so stark zu vereinfachen, dass man am Ende sofort erkennt, dass $WB \vDash Q$. Man bezeichnet diesen syntaktischen Prozess als **Ableitung**

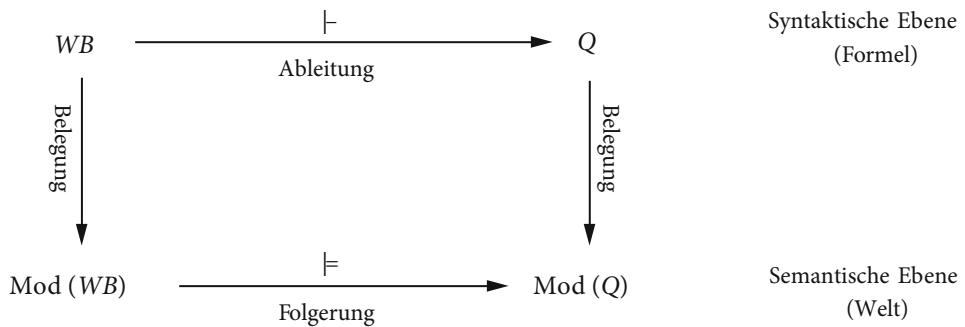


Abb. 2.1 Syntaktische Ableitung und semantische Folgerung. $\text{Mod}(X)$ steht für die Menge der Modelle einer Formel X

und schreibt $WB \vdash Q$. Solch ein syntaktisches Beweisverfahren wird **Kalkül** genannt. Um sicherzustellen, dass ein Kalkül keine Fehler macht, definieren wir zwei fundamentale Eigenschaften von Kalkülen.

Definition 2.7

Ein Kalkül heißt **korrekt**, wenn jede abgeleitete Aussage auch semantisch folgt, das heißt, wenn für Formeln WB und Q gilt

$$\text{wenn } WB \vdash Q \text{ dann } WB \vDash Q.$$

Ein Kalkül heißt **vollständig**, wenn alle semantischen Folgerungen abgeleitet werden können, das heißt, wenn für Formeln WB und Q gilt

$$\text{wenn } WB \vDash Q \text{ dann } WB \vdash Q.$$

Die Korrektheit eines Kalküls stellt also sicher, dass alle abgeleiteten Formeln tatsächlich semantische Folgerungen aus der Wissensbasis sind. Der Kalkül produziert keine „falschen Folgerungen“. Die Vollständigkeit eines Kalküls hingegen stellt sicher, dass der Kalkül nichts übersieht. Ein vollständiger Kalkül findet immer einen Beweis, wenn die zu beweisende Formel aus der Wissensbasis folgt. Ist ein Kalkül korrekt und vollständig, so sind syntaktische Ableitung und semantische Folgerung zwei äquivalente Relationen (siehe Abb. 2.1).

Um die automatischen Beweisverfahren möglichst einfach zu halten, arbeiten diese meist auf Formeln in konjunktiver Normalform.

Definition 2.8

Eine Formel ist in **konjunktiver Normalform (KNF)** genau dann, wenn sie aus einer **Konjunktion**

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

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

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

von Literalen. Ein **Literal** schließlich ist eine Variable (positives Literal) oder eine negierte Variable (negatives Literal).

Die Formel $(A \vee B \vee \neg C) \wedge (A \vee B) \wedge (\neg B \vee \neg C)$ ist in konjunktiver Normalform. Die konjunktive Normalform stellt keine Einschränkung der Formelmenge dar, denn es gilt

Satz 2.4

Jede aussagenlogische Formel lässt sich in eine äquivalente konjunktive Normalform transformieren.

Beispiel 2.1

Wir bringen $A \vee B \Rightarrow C \wedge D$ in konjunktive Normalform, indem wir die Äquivalenzen aus Satz 2.1 anwenden:

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

Zum syntaktischen Beweisen von aussagenlogischen Formeln fehlt uns nun nur noch ein Kalkül. Wir starten mit dem **Modus Ponens**, einer einfachen intuitiven Inferenzregel, die aus der Gültigkeit von A und $A \Rightarrow B$ die Ableitung von B erlaubt. Formal schreibt man dies so

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

Diese Schreibweise bedeutet, dass aus den durch Komma getrennten Formeln über der Linie die Formel(n) unter der Linie hergeleitet werden können. Modus Ponens als einzige Regel ist zwar korrekt (siehe Aufgabe 2.3), aber nicht vollständig. Fügt man weitere Regeln hinzu, so kann man einen vollständigen Kalkül erzeugen, den wir hier aber nicht betrachten wollen. Stattdessen werden wir als Alternative die **Resolutionsregel**

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

untersuchen. Die abgeleitete Klausel wird Resolvente genannt. Durch leichte Umformung erhalten wir die äquivalente Form

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

Setzt man hier A auf f , so erkennt man, dass die Resolutionsregel eine Verallgemeinerung des Modus Ponens ist. Genauso ist die Resolutionsregel anwendbar, wenn C fehlt oder wenn A und C fehlen. Im letzten Fall wird aus dem Widerspruch $B \wedge \neg B$ die leere Klausel abgeleitet (Aufgabe 2.7).

2.4 Resolution

Wir verallgemeinern nun die Resolutionsregel nochmal, indem wir Klauseln mit einer beliebigen Zahl von Literalen zulassen. Mit den Literalen $A_1, \dots, A_m, B, C_1, \dots, C_n$ lautet die **allgemeine Resolutionsregel**

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

Man nennt die Literale B und $\neg B$ komplementär. Die Resolutionsregel löscht aus den beiden Klauseln ein Paar von komplementären Literalen und vereinigt alle restlichen Literale zu einer neuen Klausel.

Um zu beweisen, dass aus einer Wissensbasis WB eine Anfrage Q folgt, wird ein Widerspruchsbeweis durchgeführt. Nach Satz 2.3 ist zu zeigen, dass aus $WB \wedge \neg Q$ ein Widerspruch ableitbar ist. Bei Formeln in konjunktiver Normalform tritt ein Widerspruch in Form von zwei Klauseln (A) und ($\neg A$) auf, die zur leeren Klausel als Resolvente führen. Dass dieses Verfahren auch wirklich wie gewünscht funktioniert, sichert uns der folgende Satz.

Damit der Kalkül auch vollständig wird, benötigen wir noch eine kleine Ergänzung, wie folgendes einfache Beispiel zeigt. Sei als Wissensbasis die Formel $(A \vee A)$ gegeben. Um per Resolution zu zeigen, dass daraus $(A \wedge A)$ ableitbar ist, muss gezeigt werden, dass aus $(A \vee A) \wedge (\neg A \vee \neg A)$ die leere Klausel ableitbar ist. Mit der Resolutionsregel allein ist

das nicht möglich. Mit der **Faktorisierung**, die es erlaubt, Kopien von Literalen in Klauseln zu löschen, wird dieses Problem beseitigt. In dem Beispiel führt zweimalige Anwendung der Faktorisierung zu $(A) \wedge (\neg A)$ und ein Resolutionsschritt zur leeren Klausel.

Satz 2.5

Der Resolutionskalkül zum Beweis der Unerfüllbarkeit von Formeln in konjunktiver Normalform ist korrekt und vollständig. Das heißt, für jede unerfüllbare Formel kann man mit Resolution die leere Klausel herleiten und umgekehrt.

Da es Aufgabe des Resolutionskalküls ist, einen Widerspruch aus $WB \wedge \neg Q$ herzuleiten, ist es ganz wichtig, dass die Wissensbasis WB **widerspruchsfrei** ist.

Definition 2.9

Eine Formel WB heißt **widerspruchsfrei** oder **konsistent**, wenn sich aus ihr kein Widerspruch, das heißt keine Formel der Form $\phi \wedge \neg\phi$, ableiten lässt.

Andernfalls lässt sich aus WB alles herleiten (siehe Aufgabe 2.8). Dies gilt nicht nur für die Resolution, sondern für viele andere Kalküle auch.

Unter den Kalkülen zum automatischen Beweisen spielt die Resolution eine herausragende Rolle. Daher wollen wir uns mit ihr etwas näher beschäftigen. Gegenüber anderen Kalkülen besitzt die Resolution nur zwei Inferenzregeln, und sie arbeitet auf Formeln in konjunktiver Normalform. Dies macht die Implementierung einfacher. Ein weiterer Vorteil gegenüber vielen Kalkülen liegt in der Reduktion der Anzahl von Möglichkeiten für die Anwendung von Inferenzregeln in jedem Schritt beim Beweisen, wodurch dann der Suchraum reduziert und die Rechenzeit verringert wird.

Wir starten mit einem einfachen Logikrätsel als Beispiel, an dem sich alle wichtigen Schritte eines Resolutionsbeweises gut zeigen lassen.

Beispiel 2.2

Das Logikrätsel Nr. 7 aus [Ber89] mit dem Titel *A charming English family* lautet:

Nachdem ich sieben Jahre lang mit glänzendem Erfolg Englisch studiert habe, muss ich zugeben, dass ich, wenn ich Engländer englisch sprechen höre, vollkommen perplex bleibe. Nun habe ich neulich, von edlen Gefühlen bewegt, drei Anhalter, Vater, Mutter und Tochter, mitgenommen, die, wie ich schnell begriff, Engländer waren und folglich nur Englisch sprachen. Bei jedem ihrer Sätze zögerte ich zwischen zwei möglichen Bedeutungen. Sie sagten mir folgendes (der zweite Sinn steht in Klammern): Der Vater: „Wir fahren bis nach Spanien (wir kommen aus Newcastle).“ Die Mutter: „Wir fahren nicht nach Spanien und kommen aus Newcastle (wir haben in Paris angehalten und fahren nicht nach Spanien).“ Die Tochter: „Wir

kommen nicht aus Newcastle (wir haben in Paris angehalten).“ What about this charming English family?

Zur Lösung derartiger Aufgaben geht man in drei Schritten vor: Formalisierung, Transformation in Normalform und Beweis. In vielen Fällen ist die Formalisierung der mit Abstand schwierigste Schritt, denn man kann dabei viele Fehler machen oder Kleinigkeiten vergessen (Daher ist das praktische Üben hier ganz wichtig. Siehe Aufgaben 2.9–2.11).

Wir verwenden hier die Variablen S für „*Wir fahren bis nach Spanien*“, N für „*Wir kommen aus Newcastle*“ und P für „*wir haben in Paris angehalten*“ und erhalten als Formalisierung der drei Aussagen von Vater, Mutter und Tochter

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

Ausklemmern von $\neg S$ in der mittleren Teilformel bringt die Formel in einem Schritt in KNF. Nummerierung der Klauseln durch tiefgestellte Indizes ergibt

$$WB \equiv (S \vee N)_1 \wedge (\neg S)_2 \wedge (P \vee N)_3 \wedge (\neg N \vee P)_4.$$

Nun starten wir den Resolutionsbeweis, vorerst noch ohne eine Anfrage Q . Ein Ausdruck der Form „Res(m,n): $\langle \text{Klausel} \rangle_k$ “ bedeutet, dass $\langle \text{Klausel} \rangle$ durch Resolution von Klausel m mit Klausel n entstanden ist und die Nummer k erhält.

$$\begin{aligned} \text{Res}(1,2) : & (N)_5 \\ \text{Res}(3,4) : & (P)_6 \\ \text{Res}(1,4) : & (S \vee P)_7 \end{aligned}$$

Die Klausel P hätten wir auch erhalten können durch Res(4,5) oder Res(2,7). Jeder weitere Resolutionsschritt würde zur Ableitung schon vorhandener Klauseln führen. Da sich die leere Klausel nicht ableiten lässt, ist damit gezeigt, dass die Wissensbasis widerspruchsfrei ist. Bis jetzt haben wir N und P abgeleitet. Um zu zeigen, dass $\neg S$ gilt, fügen wir die Klausel $(S)_8$ als negierte Anfrage zur Klauselmenge hinzu. Durch den Resolutionsschritt

$$\text{Res}(2,8) : ()_9$$

ist der Beweis geschafft. Es gilt also $\neg S \wedge N \wedge P$. Die „charming English family“ kommt offenbar aus Newcastle, hat in Paris angehalten und fährt nicht nach Spanien.

Beispiel 2.3

Das Logikrätsel Nr. 28 aus [Ber89] mit dem Titel *Der Hochsprung* lautet:

Drei junge Mädchen üben Hochsprung für die Sportprüfung im Abitur. Die Stange wurde bei 1,20 m befestigt. „Ich wette“, sagt das erste zum zweiten Mädchen, „dass mir mein Sprung

gelingen wird, wenn, und nur dann, wenn du versagst.“ Wenn das zweite junge Mädchen das gleiche zu dem dritten sagt, welches wiederum das gleiche zu dem ersten sagt, wäre es dann möglich, dass keins von den dreien seine Wette verliert?

Wir zeigen per Resolutionsbeweis, dass nicht alle drei Mädchen ihre Wette gewinnen können.

Formalisierung:

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

Behauptung: Es können nicht alle drei die Wette gewinnen:

$$Q \equiv \neg((A \Leftrightarrow \neg B) \wedge (B \Leftrightarrow \neg C) \wedge (C \Leftrightarrow \neg A))$$

Per Resolution ist nun zu zeigen, dass $\neg Q$ unerfüllbar ist.

Transformation in KNF: Wette des ersten Mädchens:

$$(A \Leftrightarrow \neg B) \equiv (A \Rightarrow \neg B) \wedge (\neg B \Rightarrow A) \equiv (\neg A \vee \neg B) \wedge (A \vee B)$$

Die Wetten der beiden anderen Mädchen werden analog transformiert, und man erhält als negierte Behauptung

$$\begin{aligned} \neg Q \equiv & (\neg A \vee \neg B)_1 \wedge (A \vee B)_2 \wedge (\neg B \vee \neg C)_3 \wedge (B \vee C)_4 \\ & \wedge (\neg C \vee \neg A)_5 \wedge (C \vee A)_6. \end{aligned}$$

Daraus wird nun mit Resolution die leere Klausel abgeleitet:

$$\begin{aligned} \text{Res}(1,6) : & (C \vee \neg B)_7 \\ \text{Res}(4,7) : & (C)_8 \\ \text{Res}(2,5) : & (B \vee \neg C)_9 \\ \text{Res}(3,9) : & (\neg C)_{10} \\ \text{Res}(8,10) : & () \end{aligned}$$

Damit ist die Behauptung gezeigt.

2.5 Hornklauseln

Eine Klausel in konjunktiver Normalform enthält positive und negative Literale und lässt sich daher darstellen in der Form

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

mit den Variablen A_1, \dots, A_m und B_1, \dots, B_n . Diese Klausel lässt sich in zwei einfachen Schritten umformen in die äquivalente Form

$$A_1 \wedge \dots \wedge A_m \Rightarrow B_1 \vee \dots \vee B_n.$$

Diese Implikation enthält als Prämisse (Voraussetzung) eine Konjunktion von Variablen und als Konklusion (Folgerung) eine Disjunktion von Variablen. Zum Beispiel ist „*Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren oder ich gehe arbeiten.*“ eine Aussage von dieser Form. Der Empfänger dieser Botschaft weiß dann immerhin, dass der Absender zum Beispiel nicht schwimmen geht. Eine wesentlich klarere Aussage wäre „*Wenn das Wetter schön ist und Schnee liegt, gehe ich Skifahren.*“ Der Empfänger weiß nun definitiv Bescheid. Daher nennt man Klauseln mit höchstens einem positiven Literal auch definite Klauseln. Diese Klauseln haben den Vorteil, dass sie nur einen Schluss zulassen und daher deutlich einfacher zu interpretieren sind. Viele Zusammenhänge lassen sich mit derartigen Klauseln beschreiben. Wir definieren daher

Definition 2.10

Klauseln mit höchstens einem positiven Literal der Formen

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

beziehungsweise (äquivalent)

$$A_1 \wedge \dots \wedge A_m \Rightarrow B \quad \text{oder} \quad A_1 \wedge \dots \wedge A_m \Rightarrow f \quad \text{oder} \quad B.$$

heißen **Hornklauseln** (nach ihrem Erfinder). Eine Klausel mit nur einem positiven Literal heißt **Fakt**. Bei Klauseln mit negativen und einem positiven Literal wird das positive Literal **Kopf** genannt.

Zum besseren Verständnis der Darstellung von Hornklauseln möge der Leser die in der Definition verwendeten Äquivalenzen herleiten (Aufgabe 2.12).

Nicht nur im täglichen Leben, sondern auch beim formalen Schließen sind Hornklauseln einfacher zu handhaben, wie man an folgendem Beispiel erkennt. Die Wissensbasis bestehet aus folgenden Klauseln (das die Klauseln verbindende „ \wedge “ wird hier und im Folgenden weggelassen):

$$\begin{aligned} & (\text{Wetter_schön})_1 \\ & (\text{Schneefall})_2 \\ & (\text{Schneefall} \Rightarrow \text{Schnee})_3 \\ & (\text{Wetter_schön} \wedge \text{Schnee} \Rightarrow \text{Skifahren})_4 \end{aligned}$$

Wollen wir nun wissen, ob *Skifahren* gilt, so lässt sich dies relativ leicht folgern. Als Inferenzregel genügt hier zum Beispiel ein leicht verallgemeinerter Modus Ponens:

$$\frac{A_1 \wedge \dots \wedge A_m, A_1 \wedge \dots \wedge A_m \Rightarrow B}{B}.$$

Der Beweis für „*Skifahren*“ hat dann folgende Gestalt ($MP(i_1, \dots, i_k)$ steht für die Anwendung des Modus Ponens auf die Klauseln i_1 bis i_k .)

$$\begin{aligned} MP(2,3) : & \quad (Schnee)_5 \\ MP(1,5,4) : & \quad (Skifahren)_6 \end{aligned}$$

Man erhält mit Modus Ponens einen vollständigen Kalkül für Formeln, die aus aussagenlogischen Hornklauseln bestehen. Bei großen Wissensbasen kann man mit Modus Ponens allerdings sehr viele unnötige Formeln ableiten, wenn man mit den falschen Klauseln startet. Besser ist es daher in vielen Fällen, einen Kalkül zu verwenden, der mit der Anfrage startet und rückwärts arbeitet, bis die Fakten erreicht sind. Solch ein Verfahren wird auch als **backward chaining** bezeichnet, im Gegensatz zum **forward chaining**, welches bei den Fakten startet und schließlich die Anfrage herleitet, so wie im obigen Beispiel mit Modus Ponens.

Für das backward chaining auf Hornklauseln wird **SLD-Resolution** verwendet. SLD steht für „*Selection rule driven linear resolution for definite clauses*“. Am obigen Beispiel, ergänzt durch die negierte Anfrage ($Skifahren \Rightarrow f$)

$$\begin{aligned} & (Wetter_schön)_1 \\ & (Schneefall)_2 \\ & (Schneefall \Rightarrow Schnee)_3 \\ & (Wetter_schön \wedge Schnee \Rightarrow Skifahren)_4 \\ & (Skifahren \Rightarrow f)_5 \end{aligned}$$

führt die SLD-Resolution, beginnend mit dieser Klausel folgende Resolutionsschritte aus

$$\begin{aligned} Res(5,4) : & \quad (Wetter_schön \wedge Schnee \Rightarrow f)_6 \\ Res(6,1) : & \quad (Schnee \Rightarrow f)_7 \\ Res(7,3) : & \quad (Schneefall \Rightarrow f)_8 \\ Res(8,2) : & \quad () \end{aligned}$$

und leitet mit der leeren Klausel einen Widerspruch her. Man erkennt hier gut die „*linear resolution*“, was bedeutet, dass immer mit der gerade aktuell hergeleiteten Klausel weitergearbeitet wird. Dies führt zu einer starken Reduktion des Suchraumes. Außerdem werden die Literale der aktuellen Klausel immer der Reihe nach in fester Reihenfolge (z.B. von links nach rechts) abgearbeitet („*Selection rule driven*“). Die Literale der aktuellen Klausel werden **Teilziele** (engl. subgoals) genannt. Die Literale der negierten Anfrage sind die **Ziele**

(engl. goals). Die Inferenzregel für einen Schritt lautet

$$\frac{A_1 \wedge \dots \wedge A_m \Rightarrow B_1, \quad B_1 \wedge B_2 \wedge \dots \wedge B_n \Rightarrow f}{A_1 \wedge \dots \wedge A_m \wedge B_2 \wedge \dots \wedge B_n \Rightarrow f}.$$

Vor Anwendung der Inferenzregel sind B_1, B_2, \dots, B_n – die aktuellen Teilziele – zu beweisen. Nach der Anwendung wird B_1 ersetzt durch die neuen Teilziele $A_1 \wedge \dots \wedge A_m$. Um zu zeigen, dass B_1 wahr ist, muss nun gezeigt werden, dass $A_1 \wedge \dots \wedge A_m$ wahr sind. Dieser Prozess setzt sich so lange fort, bis die Liste der Teilziele der aktuellen Klausel (der so genannte **goal stack**) leer ist. Damit ist ein Widerspruch gefunden. Gibt es zu einem Teilziel $\neg B_i$ keine Klausel mit einem komplementären Literal B_i als Klauselkopf, so terminiert der Beweis und es kann kein Widerspruch gefunden werden. Die Anfrage ist also nicht beweisbar.

SLD-Resolution spielt in der Praxis eine wichtige Rolle, denn Programme in der Logikprogrammiersprache Prolog bestehen aus prädikatenlogischen Hornklauseln, und ihre Abarbeitung erfolgt mittels SLD-Resolution (siehe Aufgabe 2.13, bzw. Kap. 5).

2.6 Berechenbarkeit und Komplexität

Die Wahrheitstafelmethode als einfachstes semantisches Beweisverfahren für die Aussagenlogik stellt einen Algorithmus dar, der für jede Formel nach endlicher Zeit alle Modelle bestimmt. Damit sind die Mengen der unerfüllbaren, der erfüllbaren und der allgemeingültigen Formeln entscheidbar. Die Rechenzeit der Wahrheitstafelmethode für die Erfüllbarkeit wächst im Worst-Case exponentiell mit der Zahl n der Variablen, denn die Wahrheitstabelle besitzt 2^n Zeilen. Eine Optimierung, die Methode der **Semantischen Bäume**, vermeidet die Betrachtung von Variablen, die in Klauseln nicht vorkommen, und spart daher in vielen Fällen Rechenzeit, aber im Worst-Case ist sie ebenfalls exponentiell.

Bei der Resolution wächst die Zahl der abgeleiteten Klauseln im Worst-Case exponentiell mit der Zahl der anfänglich vorhandenen Klauseln [Let03]. Für die Entscheidung zwischen den beiden Verfahren kann man daher als Faustregel angeben, dass bei vielen Klauseln mit wenigen Variablen die Wahrheitstafelmethode vorzuziehen ist und bei wenigen Klauseln mit vielen Variablen die Resolution voraussichtlich schneller zum Ziel kommen wird.

Bleibt noch die Frage, ob das Beweisen in der Aussagenlogik nicht schneller gehen kann. Gibt es bessere Algorithmen? Die Antwort lautet: Vermutlich nicht. Denn S. Cook, der Begründer der Komplexitätstheorie, hat gezeigt, dass das 3-Sat-Problem NP-vollständig ist. 3-Sat ist die Menge aller KNF-Formeln, deren Klauseln genau drei Literale haben. Damit ist klar, dass es vermutlich (modulo dem P/NP-Problem) keinen polynomiellen Algorithmus für 3-Sat und damit auch nicht allgemein geben wird. Für Hornklauseln gibt es jedoch einen Algorithmus, bei dem die Rechenzeit für das Testen der Erfüllbarkeit nur linear mit der Zahl der Literale in der Formel wächst.

2.7 Anwendungen und Grenzen

Aussagenlogische Beweiser gehören in der Digitaltechnik zum täglichen Handwerkszeug der Entwickler. Zum Beispiel die Verifikation digitaler Schaltungen oder die Generierung von Testmustern zum Test von Mikroprozessoren in der Fertigung gehören zu diesen Aufgaben. Hier kommen auch spezielle Beweisverfahren zum Einsatz, die auf binären Entscheidungsdiagrammen (engl. binary decision diagram, BDD) als Datenstruktur für aussagenlogische Formeln arbeiten.

In der KI kommt die Aussagenlogik bei einfachen Anwendungen zum Einsatz. Zum Beispiel kann ein einfaches Expertensystem durchaus mit Aussagenlogik arbeiten. Allerdings müssen die Variablen alle diskret sein, mit wenigen Werten, und es dürfen keine Querbeziehungen zwischen den Variablen bestehen. Komplexere logische Zusammenhänge können mit der Prädikatenlogik wesentlich eleganter ausgedrückt werden.

Eine sehr interessante und aktuelle Kombination von Aussagenlogik und Wahrscheinlichkeitsrechnung zur Modellierung von unsicherem Wissen ist die probabilistische Logik, die in Kap. 7 ausführlich behandelt wird. Auch die Fuzzy-Logik, welche unendlich viele Wahrheitswerte erlaubt, wird in diesem Kapitel besprochen.

2.8 Übungen

Aufgabe 2.1 ➔ Geben Sie eine Backus-Naur-Form-Grammatik für die Syntax der Aussagenlogik an.

Aufgabe 2.2 Zeigen Sie, dass folgende Formeln Tautologien sind:

- $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
- $A \Rightarrow B \Leftrightarrow \neg B \Rightarrow \neg A$
- $((A \Rightarrow B) \wedge (B \Rightarrow A)) \Leftrightarrow (A \Leftrightarrow B)$
- $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$

Aufgabe 2.3 Transformieren Sie folgende Formeln in konjunktive Normalform:

- $A \Leftrightarrow B$
- $A \wedge B \Leftrightarrow A \vee B$
- $A \wedge (A \Rightarrow B) \Rightarrow B$

Aufgabe 2.4 Überprüfen Sie die folgenden Aussagen auf Erfüllbarkeit oder Wahrheit.

- $(\text{Lotto_spielen} \wedge 6\text{_Richtige}) \Rightarrow \text{Gewinn}$
- $(\text{Lotto_spielen} \wedge 6\text{_Richtige} \wedge (6\text{_Richtige} \Rightarrow \text{Gewinn})) \Rightarrow \text{Gewinn}$
- $\neg(\neg \text{Benzin_im_Tank} \wedge (\text{Benzin_im_Tank} \vee \neg \text{Auto_startet})) \Rightarrow \neg \text{Auto_startet}$

Aufgabe 2.5 *** Programmieren Sie einen aussagenlogischen Beweiser mit der Wahrheitstafelmethode für Formeln in konjunktiver Normalform in einer Programmiersprache

Ihrer Wahl. Um einen aufwändigen Syntaxcheck der Formel zu vermeiden, können Sie die Klauseln als Listen oder Mengen von Literalen darstellen und die Formel als Liste oder Menge von Klauseln. Das Programm soll angeben, ob die Formel unerfüllbar, erfüllbar, oder wahr ist und die Zahl der unterschiedlichen Belegungen und Modelle ausgeben.

Aufgabe 2.6

- Zeigen Sie, dass der Modus Ponens eine korrekte Inferenzregel ist, indem Sie zeigen, dass $A \wedge (A \Rightarrow B) \vDash B$.
- Zeigen Sie, dass die Resolutionsregel (2.1) eine korrekte Inferenzregel ist.

Aufgabe 2.7 * Zeigen Sie unter Verwendung der Resolutionsregel, dass in der konjunktiven Normalform die leere Klausel der falschen Aussage entspricht.

Aufgabe 2.8 * Zeigen Sie, dass sich aus einer Wissensbasis, die einen Widerspruch enthält, mit Resolution jede beliebige Klausel „herleiten“ lässt.

Aufgabe 2.9 Formalisieren Sie folgende logischen Funktionen mit den logischen Operatoren und zeigen Sie, dass Ihre Formel korrekt ist. Stellen Sie das Ergebnis in KNF dar.

- Die XOR-Verknüpfung (exklusives Oder) von zwei Variablen.
- Die Aussage *mindestens zwei der drei Variablen A, B, C sind wahr*.

Aufgabe 2.10 * Lösen Sie folgenden Kriminalfall mit Hilfe eines Resolutionsbeweises: „Hatte der Verbrecher einen Komplizen, dann ist er mit dem Wagen gekommen. Der Verbrecher hatte keinen Komplizen und hatte den Schlüssel nicht, oder er hatte einen Komplizen und den Schlüssel. Der Verbrecher hatte den Schlüssel. Ist der Verbrecher mit dem Wagen gekommen, oder nicht?“

Aufgabe 2.11 Beweisen Sie mit Resolution, dass die Formel aus

- Aufgabe 2.2 d eine Tautologie ist.
- Aufgabe 2.4 c unerfüllbar ist.

Aufgabe 2.12 Zeigen Sie folgende für das Arbeiten mit Hornklauseln wichtige Äquivalenzen

- $(\neg A_1 \vee \dots \vee \neg A_m \vee B) \equiv A_1 \wedge \dots \wedge A_m \Rightarrow B$
- $(\neg A_1 \vee \dots \vee \neg A_m) \equiv A_1 \wedge \dots \wedge A_m \Rightarrow f$
- $A \equiv w \Rightarrow A$

Aufgabe 2.13 Beweisen Sie mit SLD-Resolution, dass folgende Hornklauselmenge unerfüllbar ist.

$$\begin{array}{lll}
 (A)_1 & (D)_4 & (A \wedge D \Rightarrow G)_7 \\
 (B)_2 & (E)_5 & (C \wedge F \wedge E \Rightarrow H)_8 \\
 (C)_3 & (A \wedge B \wedge C \Rightarrow F)_6 & (H \Rightarrow f)_9
 \end{array}$$

Aufgabe 2.14 => In Abschn. 2.6 heißt es „Damit ist klar, dass es vermutlich (modulo dem P/NP-Problem) keinen polynomiellen Algorithmus für 3-Sat und damit auch nicht allgemein geben wird.“ Begründen Sie das „vermutlich“ in diesem Satz näher.

Viele praktisch relevante Problemstellungen lassen sich mit der Sprache der Aussagenlogik nicht oder nur sehr umständlich formulieren, wie man an folgenden Beispielen gut erkennt.
Die Aussage

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

kann zwar direkt als die aussagenlogische Variable

„Roboter_7_befindet_sich_an_xy-Position_(35,79)“

zum Schließen mit Aussagenlogik verwendet werden, aber das Schließen mit derartigen Aussagen wird sehr umständlich. Angenommen 100 dieser Roboter können sich auf einem Gitter mit 100×100 Punkten aufhalten. Um alle Positionen aller Roboter zu beschreiben, würde man $100 \cdot 100 \cdot 100 = 1.000.000 = 10^6$ verschiedene Variablen benötigen. Erst recht schwierig wird die Definition von Relationen zwischen verschiedenen Objekten (hier Robotern). Die Relation

„Roboter A steht weiter rechts als Roboter B.“

ist semantisch letztlich nichts anderes als eine Menge von Paaren. Von den 10.000 möglichen Paaren aus x -Koordinaten sind $(100 \cdot 99)/2 = 4950$ geordnet. Zusammen mit allen 10.000 Kombinationen möglicher y -Werte beider Roboter ergeben sich zur Definition dieser Relation $100 \cdot 99 = 9900$ Formeln der Art

*Roboter_7_stehet_weiter_rechts_als_Roboter_12 \leftrightarrow
Roboter_7_befindet_sich_an_xy_Position_(35,79)
 \wedge Roboter_12_befindet_sich_an_xy_Position_(10,93) $\vee \dots$*

mit je $(10^4)^2 \cdot 0,495 = 0,495 \cdot 10^8$ Alternativen auf der rechten Seite. In der Prädikatenlogik erster Stufe definiert man hierfür ein Prädikat *Position*(*nummer*, *xPosition*, *yPosition*).

Die obige Relation muss nun nicht mehr als riesige Menge von Paaren aufgezählt werden, sondern wird abstrakt beschrieben durch eine Regel der Form

$$\forall u \forall v \text{ Steht_weiter_rechts}(u, v) \Leftrightarrow \exists x_u \exists y_u \exists x_v \exists y_v \text{ Position}(u, x_u, y_u) \wedge \text{Position}(v, x_v, y_v) \wedge x_u > x_v,$$

wobei $\forall u$ als „für alle u “ und $\exists v$ als „es gibt v “ gelesen wird.

Wir werden in diesem Kapitel Syntax und Semantik der **Prädikatenlogik erster Stufe (PL1)** definieren und aufzeigen, dass mit dieser Sprache viele Anwendungen modelliert werden können und dass es korrekte und vollständige Kalküle für diese Sprache gibt.

3.1 Syntax

Zuerst legen wir die syntaktische Struktur von Termen fest.

Definition 3.1

Sei V eine Menge von Variablen, K eine Menge von Konstanten und F eine Menge von Funktionssymbolen. Die Mengen V , K und F seien paarweise disjunkt. Die Menge der **Terme** wird rekursiv definiert:

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

Beispiele für Terme sind $f(\sin(\ln(3)), \exp(x))$ oder $g(g(g(x)))$. Um logische Beziehungen zwischen Termen herstellen zu können, bauen wir aus Termen Formeln auf.

Definition 3.2

Sei P eine Menge von Prädikatssymbolen. **Prädikatenlogische Formeln** sind wie folgt aufgebaut:

- Sind t_1, \dots, t_n Terme und p ein n -stelliges Prädikatssymbol, so ist $p(t_1, \dots, t_n)$ eine (atomare) Formel.
- Sind A und B Formeln, so sind auch $\neg A$, (A) , $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ Formeln.
- Ist x eine Variable und A eine Formel, so sind auch $\forall x A$ und $\exists x A$ Formeln. \forall ist der All- und \exists der Existenzquantor.
- $p(t_1, \dots, t_n)$ und $\neg p(t_1, \dots, t_n)$ heißen Literale.

Tab. 3.1 Beispiele für Formeln der Prädikatenlogik erster Stufe. Man beachte, dass *mutter* hier ein Funktionssymbol ist

Formel	Beschreibung
$\forall x \text{frosch}(x) \Rightarrow \text{grün}(x)$	Alle Frösche sind grün
$\forall x \text{frosch}(x) \wedge \text{braun}(x) \Rightarrow \text{groß}(x)$	Alle braunen Frösche sind groß
$\forall x \text{mag}(x, \text{kuchen})$	Jeder mag Kuchen
$\neg \forall x \text{mag}(x, \text{kuchen})$	Nicht jeder mag Kuchen
$\neg \exists x \text{mag}(x, \text{kuchen})$	Keiner mag Kuchen
$\exists x \forall y \text{mag}(y, x)$	Es gibt etwas, das jeder mag
$\exists x \forall y \text{mag}(x, y)$	Es gibt jemanden, der alles mag
$\forall x \exists y \text{mag}(y, x)$	Jedes Ding wird von jemandem geliebt
$\forall x \exists y \text{mag}(x, y)$	Jeder mag etwas
$\forall x \text{kunde}(x) \Rightarrow \text{mag}(\text{bob}, x)$	Bob mag jeden Kunden
$\exists x \text{kunde}(x) \wedge \text{mag}(x, \text{bob})$	Es gibt einen Kunden, der Bob mag
$\exists x \text{bäcker}(x) \wedge \forall y \text{kunde}(y) \Rightarrow \text{mag}(x, y)$	Es gibt einen Bäcker, der all seine Kunden mag
$\forall x \text{älter}(\text{mutter}(x), x)$	Jede Mutter ist älter als ihr Kind
$\forall x \text{älter}(\text{mutter}(\text{mutter}(x)), x)$	Jede Großmutter ist älter als das Kind ihrer Tochter
$\forall x \forall y \forall z \text{rel}(x, y) \wedge \text{rel}(y, z) \Rightarrow \text{rel}$ ist eine transitive Relation $\text{rel}(x, z)$	rel ist eine transitive Relation

- Formeln, bei denen jede Variable im Wirkungsbereich eines Quantors ist, heißen **Aussagen** oder **geschlossene Formeln**. Variablen, die nicht im Wirkungsbereich eines Quantors sind, heißen **freie Variablen**.
- Die Definitionen 2.8 (KNF) und 2.10 (Hornklausel) gelten für Formeln aus prädikatenlogischen Literalen analog.

In Tab. 3.1 sind einige Beispiele für PL1-Formeln und jeweils eine intuitive Interpretation angegeben.

3.2 Semantik

In der Aussagenlogik wird bei einer Belegung jeder Variablen direkt ein Wahrheitswert zugeordnet. In der Prädikatenlogik wird die Bedeutung von Formeln rekursiv über den Formelaufbau definiert, indem wir zuerst den Konstanten, Variablen und Funktionssymbolen Objekte in der realen Welt zuordnen.

Definition 3.3

Eine **Belegung** \mathbb{B} oder **Interpretation** ist definiert durch

- Eine Abbildung von der Menge der Konstanten und Variablen $K \cup V$ auf eine Menge W von Namen von Objekten aus der Welt.
- Eine Abbildung von der Menge der Funktionssymbole auf die Menge der Funktionen der Welt. Jedem n -stelligen Funktionssymbol wird eine n -stellige Funktion zugeordnet.
- Eine Abbildung von der Menge der Prädikatssymbole auf die Menge der Relationen der Welt. Jedem n -stelligen Prädikatssymbol wird eine n -stellige Relation zugeordnet.

Beispiel 3.1

Seien c_1, c_2, c_3 Konstanten, „*plus*“ ein zweistelliges Funktionssymbol und „*gr*“ ein zweistelliges Prädikatssymbol. Die Wahrheit der Formel

$$F \equiv gr(\textit{plus}(c_1, c_3), c_2)$$

hängt von der Belegung \mathbb{B} ab. Wir wählen zuerst folgende naheliegende Belegung der Konstanten, der Funktion und des Prädikates in den natürlichen Zahlen:

$$\mathbb{B}_1 : c_1 \mapsto 1, c_2 \mapsto 2, c_3 \mapsto 3, \quad \textit{plus} \mapsto +, \quad \textit{gr} \mapsto > .$$

Damit wird die Formel abgebildet auf

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

Die Größer-Relation auf der Menge $\{1, 2, 3, 4\}$ ist die Menge aller Paare (x, y) von Zahlen mit $x > y$, das heißt die Menge $G = \{(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)\}$. Da $(4, 2) \in G$, ist die Formel F unter der Belegung \mathbb{B}_1 wahr. Wählen wir jedoch die Belegung

$$\mathbb{B}_2 : c_1 \mapsto 2, c_2 \mapsto 3, c_3 \mapsto 1, \quad \textit{plus} \mapsto -, \quad \textit{gr} \mapsto >,$$

so erhalten wir

$$2 - 1 > 3, \quad \text{bzw.} \quad 1 > 3.$$

Das Paar $(1, 3)$ ist nicht in G enthalten. Daher ist die Formel F unter der Belegung \mathbb{B}_2 falsch. Die Wahrheit einer Formel in PL1 hängt also von der Belegung ab. Nach diesem Vorgriff nun die Definition des Wahrheitsbegriffes.

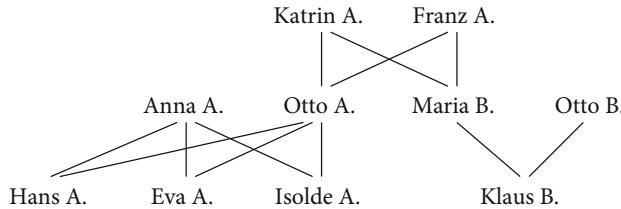


Abb. 3.1 Ein Stammbaum. Das von Klaus B. nach oben zu Maria B. und Otto B. abgehende Paar von Kanten stellt das Element (Klaus B., Maria B., Otto B.) der Kind-Relation dar

Definition 3.4

- Eine atomare Formel $p(t_1, \dots, t_n)$ ist **wahr** unter der Belegung \mathbb{B} , wenn nach Belegung und Auswertung aller Terme t_1, \dots, t_n und Belegung des Prädikates p durch die n -stellige Relation r gilt

$$(\mathbb{B}(t_1), \dots, \mathbb{B}(t_n)) \in r.$$

- Die Wahrheit von quantorenfreien Formeln ergibt sich aus der Wahrheit der atomaren Formeln – wie in der Aussagenlogik – über die in Tab. 2.1 definierte Semantik der logischen Operatoren.
- Eine Formel $\forall x F$ ist wahr unter der Belegung \mathbb{B} genau dann, wenn sie bei beliebiger Änderung der Belegung für die Variable x (und nur für diese) wahr ist.
- Eine Formel $\exists x F$ ist wahr unter der Belegung \mathbb{B} genau dann, wenn es für die Variable x eine Belegung gibt, welche die Formel wahr macht.

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

Die Sätze 2.2 (Deduktionstheorem) und 2.3 (Widerspruchsbeweis) gelten analog in leicht modifizierter Form auch für PL1.

Satz 3.1

Seien A, B prädikatenlogische Formeln und A enthalte keine freien Variablen. Dann gilt $A \vDash B$ genau dann wenn $A \Rightarrow B$ eine Tautologie ist.

Außerdem gilt $A \vDash B$ genau dann wenn $A \wedge \neg B$ unerfüllbar ist.

Beispiel 3.2

Der in Abb. 3.1 angegebene Stammbaum stellt (auf der semantischen Ebene) die Relation

$$\text{Kind} = \{(\text{Otto A., Katrin A., Franz A.}, (\text{Maria B., Katrin A., Franz A.}), \\ (\text{Hans A., Anna A., Otto A.}), (\text{Eva A., Anna A., Otto A.}), \\ (\text{Isolde A., Anna A., Otto A.}), (\text{Klaus B., Maria B., Otto B.})\}$$

grafisch dar. Zum Beispiel steht das Tripel (Otto A., Katrin A., Franz A.) für die Aussage „Otto A. ist Kind von Katrin A. und Franz A.“. Aus den Namen lesen wir die einstellige Relation

$$\text{Weiblich} = \{\text{Katrin A., Anna A., Maria B., Eva A., Isolde A.}\}$$

der weiblichen Personen ab. Wir wollen nun Formeln über Verwandtschaftsbeziehungen aufstellen. Zuerst definieren wir ein dreistelliges Prädikat $\text{kind}(x, y, z)$ mit der Semantik

$$\mathbb{B}(\text{kind}(x, y, z)) = w \equiv (\mathbb{B}(x), \mathbb{B}(y), \mathbb{B}(z)) \in \text{Kind}.$$

Unter der Belegung $\mathbb{B}(\text{otto}) = \text{Otto A.}$, $\mathbb{B}(\text{eva}) = \text{Eva A.}$, $\mathbb{B}(\text{anna}) = \text{Anna A.}$ ist also $\text{kind}(\text{eva}, \text{anna}, \text{otto})$ wahr. Damit auch $\text{kind}(\text{eva}, \text{otto}, \text{anna})$ gilt, fordern wir mit

$$\forall x \forall y \forall z \text{kind}(x, y, z) \Leftrightarrow \text{kind}(x, z, y),$$

die Symmetrie des Prädikates kind in den letzten beiden Argumenten. Für weitere Definitionen verweisen wir auf Aufgabe 3.1 und definieren nun das Prädikat nachkomme rekursiv als

$$\forall x \forall y \text{nachkomme}(x, y) \Leftrightarrow \exists z \text{kind}(x, y, z) \vee \\ (\exists u \exists v \text{kind}(x, u, v) \wedge \text{nachkomme}(u, y)).$$

Jetzt bauen wir eine kleine Wissensbasis aus Regeln und Fakten auf. Es sei

$$\begin{aligned} WB \equiv & \text{weiblich(katrin)} \wedge \text{weiblich(anna)} \wedge \text{weiblich(maria)} \\ & \wedge \text{weiblich(eva)} \wedge \text{weiblich(isolde)} \\ & \wedge \text{kind(otto,katrin,franz)} \wedge \text{kind(maria,katrin,franz)} \\ & \wedge \text{kind(eva,anna,otto)} \wedge \text{kind(hans,anna,otto)} \\ & \wedge \text{kind(isolde,anna,otto)} \wedge \text{kind(klaus,maria,ottob)} \\ & \wedge (\forall x \forall y \forall z \text{kind}(x, y, z) \Rightarrow \text{kind}(x, z, y)) \\ & \wedge (\forall x \forall y \text{nachkomme}(x, y) \Leftrightarrow \exists z \text{kind}(x, y, z) \vee \\ & (\exists u \exists v \text{kind}(x, u, v) \wedge \text{nachkomme}(u, y))). \end{aligned}$$

Wir können uns nun zum Beispiel fragen, ob aus dieser Wissensbasis die Aussagen $\text{kind}(\text{eva}, \text{otto}, \text{anna})$ oder $\text{nachkomme}(\text{eva}, \text{franz})$ ableitbar sind. Dazu benötigen wir einen Kalkül.

3.2.1 Gleichheit

Um Terme vergleichen zu können ist die Gleichheit eine ganz wichtige Relation in der Prädikatenlogik. Die Gleichheit von Termen in der Mathematik ist eine Äquivalenzrelation, das heißt sie ist reflexiv, symmetrisch und transitiv. Will man die Gleichheit in Formeln verwenden, so muss man diese drei Eigenschaften als Axiome mit in die Wissensbasis aufnehmen oder man integriert die Gleichheit in den Kalkül. Wir gehen hier den einfacheren ersten Weg und definieren ein Prädikat „=“, das, wie in der Mathematik üblich, abweichend von Definition 3.2 in Infix-Notation verwendet wird. (Eine Gleichung $x = y$ könnte natürlich auch in der Form $eq(x, y)$ geschrieben werden.) Die Gleichheitsaxiome haben damit folgende Gestalt

$$\begin{array}{ll} \forall x \quad x = x & (\text{Reflexivität}) \\ \forall x \forall y \quad x = y \Rightarrow y = x & (\text{Symmetrie}) \\ \forall x \forall y \forall z \quad x = y \wedge y = z \Rightarrow x = z & (\text{Transitivität}) \end{array} \quad (3.1)$$

Um die semantische Eindeutigkeit von Funktionen zu garantieren, wird außerdem für jedes Funktionssymbol

$$\forall x \forall y \quad x = y \Rightarrow f(x) = f(y) \quad (\text{Substitutionsaxiom}) \quad (3.2)$$

gefordert. Analog fordert man für alle Prädikatssymbole

$$\forall x \forall y \quad x = y \Rightarrow p(x) \Leftrightarrow p(y) \quad (\text{Substitutionsaxiom}) \quad (3.3)$$

Ähnlich lassen sich andere mathematische Relationen wie zum Beispiel die „<“-Relation formalisieren (Aufgabe 3.4).

Oft muss eine Variable durch einen Term ersetzt werden. Um dies korrekt durchzuführen und einfach zu beschreiben, definieren wir

Definition 3.5

Man schreibt $\varphi[x/t]$ für die Formel, die entsteht, wenn man in φ jedes freie Vorkommen der Variable x durch den Term t ersetzt. Hierbei dürfen in dem Term t keine Variablen vorkommen, über die in φ quantifiziert wurde. Gegebenenfalls müssen Variablen umbenannt werden, um dies sicherzustellen.

Beispiel 3.3

Wird in der Formel $\forall x \ x = y$ die freie Variable y durch den Term $x + 1$ ersetzt, so ergibt sich $\forall x \ x = x + 1$. Bei korrekter Ersetzung erhält man die Formel $\forall x \ x = y + 1$ mit einer ganz anderen Semantik.

3.3 Quantoren und Normalformen

Nach Definition 3.4 ist eine Formel $\forall x p(x)$ genau dann wahr, wenn sie für jede Belegung der Variablen x wahr ist. Man könnte also statt dem Quantor auch schreiben $p(a_1) \wedge \dots \wedge p(a_n)$ für alle Konstanten $a_1 \dots a_n$ aus K . Für $\exists x p(x)$ könnte man $p(a_1) \vee \dots \vee p(a_n)$ schreiben. Mit den deMorgan-Regeln folgt daraus

$$\forall x \varphi \equiv \neg \exists x \neg \varphi.$$

All- und Existenzquantor sind über diese Äquivalenz also gegenseitig ersetzbar.

Beispiel 3.4

Die Aussage „Jeder will geliebt werden“ ist äquivalent zur Aussage „Keiner will nicht geliebt werden“.

Die Quantoren sind ein wichtiger Bestandteil der Ausdrucksstärke der Prädikatenlogik. Für die automatische Inferenz in der KI sind sie jedoch störend, denn sie machen die Struktur der Formeln komplexer und erhöhen daher die Zahl der in jedem Schritt eines Beweises anwendbarer Inferenzregeln. Deshalb ist unser nächstes Ziel, zu jeder prädikatenlogischen Formel eine äquivalente Formel mit möglichst wenigen Quantoren in einer standardisierten Normalform zu finden. In einem ersten Schritt bringen wir alle Quantoren ganz an den Anfang der Formel und definieren daher

Definition 3.6

Eine prädikatenlogische Formel φ ist in **pränexer Normalform**, wenn gilt

- $\varphi = Q_1 x_1 \dots Q_n x_n \psi$.
- ψ ist eine quantorenfreie Formel.
- $Q_i \in \{\forall, \exists\}$ für $i = 1, \dots, n$.

Vorsicht ist geboten, wenn eine quantifizierte Variable auch außerhalb des Wirkungsbereichs ihres Quantors auftritt, wie zum Beispiel x in

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

Hier muss eine der beiden Variablen umbenannt werden, und in

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

lässt sich der Quantor dann leicht nach vorne bringen, und wir erhalten die zur Ausgangsformel äquivalente Formel

$$\forall x \exists y p(x) \Rightarrow q(y).$$

Will man nun aber in

$$(\forall x p(x)) \Rightarrow \exists y q(y) \quad (3.4)$$

die Quantoren korrekt nach vorne bringen, so schreibt man die Formel zuerst um in die äquivalente Form

$$\neg(\forall x p(x)) \vee \exists y q(y).$$

Der erste Allquantor dreht sich nun zu

$$(\exists x \neg p(x)) \vee \exists y q(y)$$

und erst jetzt dürfen die beiden Quantoren nach vorne gezogen werden zu

$$\exists x \exists y \neg p(x) \vee q(y),$$

was wiederum äquivalent ist zu

$$\exists x \exists y p(x) \Rightarrow q(y).$$

Man erkennt daran, dass man in (3.4) nicht einfach beide Quantoren nach vorne ziehen darf. Vielmehr muss man vorher die Implikationen beseitigen, so dass keine Negationen mehr vor Quantoren stehen. Allgemein gilt, dass man die Quantoren erst dann nach außen ziehen darf, wenn Negationen nur noch direkt vor atomaren Teilformeln stehen.

Beispiel 3.5

Die aus der Analysis bekannte Konvergenz einer Folge $(a_n)_{n \in \mathbb{N}}$ gegen einen Grenzwert a ist definiert durch

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

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

$$\forall \varepsilon (\text{gr}(\varepsilon, 0) \Rightarrow \exists n_0 (\text{el}(n_0, \mathbb{N}) \Rightarrow \forall n (\text{gr}(n, n_0) \Rightarrow \text{gr}(\varepsilon, \text{abs}(\text{minus}(a(n), a)))))), \quad (3.5)$$

die offenbar nicht in pränexer Normalform ist. Da die Variablen der inneren Quantoren $\exists n_0$ und $\forall n$ nicht links vom jeweiligen Quantor vorkommen, müssen keine Variablen umbenannt werden. Wir eliminieren zunächst die Implikationen und erhalten

$$\forall \varepsilon (\neg \text{gr}(\varepsilon, 0) \vee \exists n_0 (\neg \text{el}(n_0, \mathbb{N}) \vee \forall n (\neg \text{gr}(n, n_0) \vee \text{gr}(\varepsilon, \text{abs}(\text{minus}(a(n), a)))))).$$

Da alle Negationen vor atomaren Formeln stehen, bringen wir die Quantoren nach vorne, streichen die nun überflüssigen Klammern und es entsteht mit

$$\forall \varepsilon \exists n_0 \forall n (\neg gr(\varepsilon, 0) \vee \neg el(n_0, \mathbb{N}) \vee \neg gr(n, n_0) \vee gr(\varepsilon, abs(minus(a(n), a))))$$

eine quantifizierte Klausel in konjunktiver Normalform.

Die transformierte Formel ist äquivalent zur Ausgangsformel. Dass diese Transformation immer möglich ist, garantiert der

Satz 3.2

Jede prädikatenlogische Formel lässt sich in eine äquivalente Formel in pränexer Normalform transformieren.

Man kann auch noch alle Existenzquantoren eliminieren. Allerdings ist die aus der so genannten **Skolemisierung** resultierende Formel nicht mehr semantisch äquivalent zur Ausgangsformel. Die Erfüllbarkeit bleibt jedoch erhalten. In vielen Fällen, insbesondere wenn man wie bei der Resolution die Unerfüllbarkeit von $WB \wedge \neg Q$ zeigen will, ist dies jedoch ausreichend. Folgende Formel in pränexer Normalform soll nun skolemisiert werden:

$$\forall x_1 \forall x_2 \exists y_1 \forall x_3 \exists y_2 p(f(x_1), x_2, y_1) \vee q(y_1, x_3, y_2).$$

Da offenbar die Variable y_1 von x_1 und x_2 abhängt, wird jedes Vorkommen von y_1 durch eine Skolemfunktion $g(x_1, x_2)$ ersetzt. Wichtig ist, dass g ein neues, in der Formel noch nicht vorkommendes Funktionssymbol ist. Wir erhalten

$$\forall x_1 \forall x_2 \forall x_3 \exists y_2 p(f(x_1), x_2, g(x_1, x_2)) \vee q(g(x_1, x_2), x_3, y_2)$$

und ersetzen nun analog y_2 durch $h(x_1, x_2, x_3)$, was zu

$$\forall x_1 \forall x_2 \forall x_3 p(f(x_1), x_2, g(x_1, x_2)) \vee q(g(x_1, x_2), x_3, h(x_1, x_2, x_3))$$

führt. Da nun alle Variablen allquantifiziert sind, können die Allquantoren weggelassen werden mit dem Resultat

$$p(f(x_1), x_2, g(x_1, x_2)) \vee q(g(x_1, x_2), x_3, h(x_1, x_2, x_3)).$$

Nun können wir in (3.5) den Existenzquantor (und damit auch die Allquantoren) durch die Einführung der Skolemfunktion $n_0(\varepsilon)$ eliminieren. Die skolemisierte pränexe und konjunktive Normalform von (3.5) lautet damit

$$\neg gr(\varepsilon, 0) \vee \neg el(n_0(\varepsilon), \mathbb{N}) \vee \neg gr(n, n_0(\varepsilon)) \vee gr(\varepsilon, abs(minus(a(n), a))).$$

Durch den Wegfall der Variable n_0 kann die Skolemfunktion den Namen n_0 erhalten.

NORMALFORMTRANSFORMATION(*Formel*)

1. Transformation in pränexe Normalform:

Transformation in konjunktive Normalform (Satz 2.1):

Äquivalenzen eliminieren.

Implikationen eliminieren.

Wiederholte Anwendung von deMorgan-Regeln und Distributivgesetzen.

Gegebenenfalls Variablen umbenennen.

Alle Quantoren nach außen ziehen.

2. Skolemisierung:

Existenzquantifizierte Variablen durch neue Skolemfunktionen ersetzen.

Resultierende Allquantoren löschen.

Abb. 3.2 Transformation prädikatenlogischer Formeln in konjunktive Normalform

Bei der Skolemisierung einer Formel in pränexer Normalform werden von außen nach innen alle Existenzquantoren eliminiert, indem eine Formel der Form $\forall x_1 \dots \forall x_n \exists y \varphi$ durch $\forall x_1 \dots \forall x_n \varphi[y/f(x_1, \dots, x_n)]$ ersetzt wird, wobei die Funktion f in φ nicht vorkommen darf. Steht ein Existenzquantor ganz außen, wie in $\exists y p(y)$, so wird y durch eine Konstante (d. h. ein nullstelliges Funktionssymbol) ersetzt.

Dieses Verfahren der Transformation einer Formel in konjunktive Normalform lässt sich zu dem in Abb. 3.2 dargestellten Programmschema zusammenfassen. Die Skolemisierung hat polynomielle Laufzeit in der Zahl der Literale. Bei der Transformation in Normalform kann die Zahl der Literale im Worst-Case exponentiell anwachsen, was zu exponentieller Rechenzeit und exponentiellem Speicherplatzbedarf führen kann. Grund hierfür ist die wiederholte Anwendung der Distributivgesetze. Das eigentliche Problem, das durch eine große Zahl an Klauseln entsteht, ist aber die kombinatorische Explosion des Suchraums für einen anschließenden Resolutionsbeweis. Es gibt jedoch einen optimierten Transformationsalgorithmus, der nur polynomiell viele Literale erzeugt [Ede91].

3.4 Beweiskalküle

Für das Schließen in der Prädikatenlogik wurden verschiedene Kalküle des natürlichen Schließens wie zum Beispiel der Gentzenkalkül oder der Sequenzenkalkül entwickelt. Wie der Name schon sagt, sind diese Kalküle für die Anwendung durch Menschen gedacht, da die Inferenzregeln mehr oder weniger intuitiv sind und die Kalküle auf beliebigen PL1-Formeln arbeiten. Wir werden uns aber im nächsten Abschnitt hauptsächlich auf den Resolutionskalkül als für die Praxis wichtigsten effizient automatisierbaren Kalkül für Formeln in konjunktiver Normalform konzentrieren. Hier wollen wir nur anhand von Beispiel 3.2

Tab. 3.2 Einfacher Beweis mit Modus Ponens und Quantorenelimination

WB:	1	$kind(eva, anna, otto)$
WB:	2	$\forall x \forall y \forall z kind(x, y, z) \Rightarrow kind(x, z, y)$
$\forall E(2) : x/eva, y/anna, z/otto$	3	$kind(eva, anna, otto) \Rightarrow kind(eva, otto, anna)$
$MP(1,3)$	4	$kind(eva, otto, anna)$

einen ganz kleinen „natürlichen“ Beweis angeben. Wir verwenden hier die Inferenzregeln

$$\frac{A, A \Rightarrow B}{B} \quad (\text{Modus Ponens, MP}) \quad \text{und} \quad \frac{\forall x A}{A[x/t]} \quad (\forall\text{-Elimination, } \forall E).$$

Der Modus Ponens ist schon aus der Aussagenlogik bekannt. Bei der Elimination des Allquantors ist zu beachten, dass die quantifizierte Variable x ersetzt werden muss durch einen Grundterm t , das heißt einen Term, der keine Variablen enthält. Der Beweis für $kind(eva, otto, anna)$ aus einer entsprechend reduzierten Wissensbasis ist in Tab. 3.2 dargestellte.

In den Zeilen 1 und 2 sind die beiden Formeln der reduzierten Wissensbasis aufgelistet. In Zeile 3 werden die Allquantoren aus Zeile 2 eliminiert, und in Zeile 4 wird mit Modus Ponens die Behauptung hergeleitet.

Der aus den beiden angegebenen Inferenzregeln bestehende Kalkül ist nicht vollständig. Er kann jedoch durch Hinzufügen weiterer Inferenzregeln zu einem vollständigen Verfahren ergänzt werden [Rau96]. Diese nicht triviale Tatsache ist für die Mathematik und die KI von fundamentaler Bedeutung. Der österreichische Logiker Kurt Gödel bewies 1931 [Göd31a]

Satz 3.3 (Gödelscher Vollständigkeitssatz)

Die Prädikatenlogik erster Stufe ist vollständig. Das heißt, es gibt einen Kalkül, mit dem sich jede Aussage φ , die aus einer Wissensbasis WB folgt, beweisen lässt. Wenn $WB \vDash \varphi$, dann gilt $WB \vdash \varphi$.

Jede wahre Aussage der Prädikatenlogik erster Stufe ist damit beweisbar. Gilt aber auch die Umkehrung? Ist alles, was wir syntaktisch herleiten können, auch wahr? Die Antwort lautet „ja“:

Satz 3.4 (Korrektheit)

Es gibt Kalküle, mit denen sich nur wahre Aussagen beweisen lassen. Das heißt, wenn $WB \vdash \varphi$ gilt, dann auch $WB \vDash \varphi$.

Tatsächlich sind fast alle bekannten Kalküle korrekt, denn das Arbeiten mit inkorrekten Beweisverfahren macht wenig Sinn. Beweisbarkeit und semantische Folgerung sind also äquivalente Begriffe, sofern ein Kalkül benutzt wird, der korrekt und vollständig ist. Dadurch wird die Prädikatenlogik erster Stufe zu einem starken Werkzeug für die Mathematik und die KI. Die schon erwähnten Kalküle des natürlichen Schließens sind allerdings für die Automatisierung nicht geeignet. Erst der im Jahr 1965 vorgestellte Resolutionskalkül, der im Wesentlichen mit nur einer einfachen Inferenzregel arbeitet, ermöglichte den Bau von leistungsfähigen automatischen Theorembeweisern, die dann später auch als Inferenzmaschinen für Expertensysteme eingesetzt wurden.

3.5 Resolution

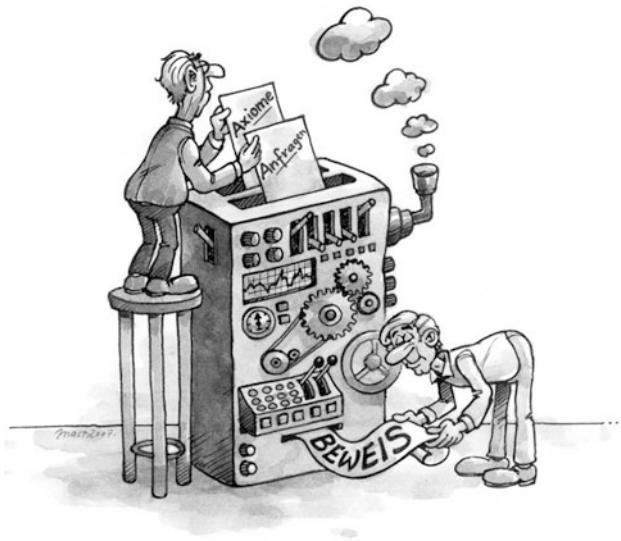
Tatsächlich löste der korrekte und vollständige Resolutionskalkül während den 70er Jahren des 20. Jahrhunderts eine regelrechte Logikeuphorie in der KI aus. Viele Wissenschaftler glaubten, man könne fast alle Aufgaben der Wissensrepräsentation und des Schließens in PL1 formulieren und dann mit einem automatischen Beweiser lösen. Die Prädikatenlogik als mächtige ausdrucksstarke Sprache zusammen mit einem vollständigen Beweiskalkül schien die universelle intelligente Maschine zur Repräsentation von Wissen und zur Lösung vieler schwieriger Probleme zu sein (Abb. 3.3).

Gibt man in solch eine Logikmaschine eine Axiomenmenge, das heißt eine Wissensbasis, und eine Anfrage ein, so sucht die Maschine einen Beweis und gibt diesen – so einer existiert und gefunden wird – dann aus. Der Gödelsche Vollständigkeitssatz als Fundament und die Arbeiten von Herbrand waren Anlass, die Mechanisierung der Logik mit großem Aufwand voranzutreiben. Die Vision einer Maschine, die zu einer beliebigen widerspruchsfreien Wissensbasis der PL1 jede wahre Anfrage beweisen kann, war sehr verlockend. Entsprechend wurden bis heute viele Beweiskalküle für PL1 entwickelt und in Form von Theorembeweisern realisiert. Wir werden hier beispielhaft den historisch wichtigen und weit verbreiteten Resolutionskalkül beschreiben und aufzeigen, was er zu leisten im Stande ist. Der Grund für die Auswahl der Resolution als Beispiel für einen Beweiskalkül in diesem Buch ist, wie gesagt, die historische und didaktische Bedeutung. Heute stellt die Resolution bei den Hochleistungsbeweisern einen unter vielen Kalkülen dar.

Wir starten, indem wir versuchen, den Beweis aus Tab. 3.2 zu Beispiel 3.2 in einen Resolutionsbeweis umzubauen. Zuerst werden die Formeln in konjunktive Normalform umgewandelt und die negierte Anfrage $\neg Q \equiv \neg \text{kind}(\text{eva}, \text{otto}, \text{anna})$ zur Wissensbasis hinzugefügt, was

$$\begin{aligned} WB \wedge \neg Q \equiv & (\text{kind}(\text{eva}, \text{anna}, \text{otto}))_1 \wedge \\ & (\neg \text{kind}(x, y, z) \vee \text{kind}(x, z, y))_2 \wedge \\ & (\neg \text{kind}(\text{eva}, \text{otto}, \text{anna}))_3 \end{aligned}$$

Abb. 3.3 Die universelle Logikmaschine



ergibt. Der Beweis könnte dann etwa so aussehen

$$\begin{aligned}
 (2) \quad & x/eva, y/anna, z/otto : (-\text{kind}(eva, anna, otto)) \vee \text{kind}(eva, otto, anna))_4 \\
 & \text{Res}(3, 4) : (-\text{kind}(eva, anna, otto))_5 \\
 & \text{Res}(1, 5) : ()_6,
 \end{aligned}$$

wobei im ersten Schritt die Variablen x, y, z durch Konstanten ersetzt werden. Dann folgen zwei Resolutionsschritte unter Verwendung der allgemeinen Resolutionsregel aus (2.2), die aus der Aussagenlogik unverändert übernommen wurde.

Etwas komplexer wird der Sachverhalt in folgendem Beispiel. Wir nehmen an, *jeder kennt seine eigene Mutter* und fragen uns, ob *Hans* jemanden kennt. Mit dem Funktionssymbol „mutter“ und dem Prädikat „kennt“ gilt es, einen Widerspruch aus

$$(\text{kennt}(x, \text{mutter}(x)))_1 \wedge (\neg \text{kennt}(\text{hans}, y))_2$$

abzuleiten. Durch die Ersetzung x/hans , $y/\text{mutter}(\text{hans})$ erhalten wir das widersprüchliche Klauselpaar

$$(\text{kennt}(\text{hans}, \text{mutter}(\text{hans})))_1 \wedge (\neg \text{kennt}(\text{hans}, \text{mutter}(\text{hans})))_2.$$

Diesen Ersetzungsschritt nennt man **Unifikation**. Die beiden Literale sind komplementär, das heißt, sie sind bis auf das Vorzeichen gleich. Mit einem Resolutionsschritt ist nun die leere Klausel ableitbar, womit gezeigt ist, dass Hans jemanden (seine Mutter) kennt. Wir definieren

Definition 3.7

Zwei Literale heißen **unifizierbar**, wenn es eine Ersetzung σ für alle Variablen gibt, welche die Literale gleich macht. Solch ein σ wird **Unifikator** genannt. Ein Unifikator heißt allgemeiner Unifikator (engl. most general unifier (MGU)), wenn sich aus ihm alle anderen Unifikatoren durch Ersetzung von Variablen ergeben.

Beispiel 3.6

Die Literale $p(f(g(x)), y, z)$ und $p(u, u, f(u))$ sollen unifiziert werden. Einige Unifikatoren sind

$$\begin{array}{llll} \sigma_1 : & y/f(g(x)), & z/f(f(g(x))), & u/f(g(x)), \\ \sigma_2 : & x/h(v), & y/f(g(h(v))), & z/f(f(g(h(v)))), & u/f(g(h(v))) \\ \sigma_3 : & x/h(h(v)), & y/f(g(h(h(v)))), & z/f(f(g(h(h(v))))), & u/f(g(h(h(v)))) \\ \sigma_4 : & x/h(a), & y/f(g(h(a))), & z/f(f(g(h(a)))), & u/f(g(h(a))) \\ \sigma_5 : & x/a, & y/f(g(a)), & z/f(f(g(a))), & u/f(g(a)) \end{array}$$

wobei σ_1 der allgemeinste Unifikator ist. Die anderen Unifikatoren ergeben sich aus σ_1 durch die Ersetzungen $x/h(v)$, $x/h(h(v))$, $x/h(a)$, x/a .

An diesem Beispiel erkennt man gut, dass bei der Unifikation von Literalen die Prädikatsymbole wie Funktionssymbole behandelt werden können. Das heißt, das Literal wird wie ein Term behandelt. Implementierungen von Unifikationsalgorithmen arbeiten die Argumente von Funktionen sequentiell ab. Terme werden rekursiv über die Termstruktur unifiziert. Die einfachsten Unifikationsalgorithmen sind in den meisten Fällen sehr schnell. Im Worst-Case jedoch kann die Rechenzeit exponentiell mit der Termgröße wachsen. Da in automatischen Beweisern die überwiegende Zahl aller Unifikationsversuche scheitern oder sehr einfach sind, wirkt sich die Worst-Case-Komplexität meist nicht dramatisch aus. Die schnellsten Unifikationsalgorithmen haben auch im Worst-Case fast lineare Komplexität [Bib92].

Nun können wir die allgemeine Resolutionsregel für Prädikatenlogik angeben:

Definition 3.8

Die Resolutionsregel für zwei Klauseln in konjunktiver Normalform lautet

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

wobei σ der MGU von B und B' ist.

Satz 3.5

Die Resolutionsregel ist korrekt, das heißt, die Resolvente folgt semantisch aus den beiden Vaterklauseln.

Zur Vollständigkeit fehlt aber noch eine kleine Ergänzung, wie uns das folgende Beispiel zeigt.

Beispiel 3.7

Die bekannte Russellsche Antinomie lautet „*Es gibt einen Barbier, der alle Menschen rasiert, die sich nicht selbst rasieren.*“ Diese Aussage ist widersprüchlich, das heißt sie ist unerfüllbar. Dies wollen wir mit Resolution zeigen. In PL1 formalisiert lautet die Antinomie

$$\forall x \text{rasiert}(\text{barbier}, x) \Leftrightarrow \neg \text{rasiert}(x, x)$$

und in Klauselform transformiert ergibt sich (siehe Aufgabe 3.6)

$$(\neg \text{rasiert}(\text{barbier}, x) \vee \neg \text{rasiert}(x, x))_1 \wedge (\text{rasiert}(\text{barbier}, x) \vee \text{rasiert}(x, x))_2. \quad (3.7)$$

Aus diesen beiden Klauseln kann man mit Resolution mehrere Tautologien ableiten, aber keinen Widerspruch. Also ist die Resolution nicht vollständig. Man benötigt noch eine weitere Inferenzregel.

Definition 3.9

Die **Faktorisierung** einer Klausel erfolgt durch

$$\frac{(A_1 \vee A_2 \vee \dots \vee A_n) \quad \sigma(A_1) = \sigma(A_2)}{(\sigma(A_2) \vee \dots \vee \sigma(A_n))}, \quad (3.8)$$

wobei σ der MGU von A_1 und A_2 ist.

Nun lässt sich aus (3.7) ein Widerspruch ableiten

$$\begin{aligned} \text{Fak}(1, \sigma : x/\text{barbier}) &: (\neg \text{rasiert}(\text{barbier}, \text{barbier}))_3 \\ \text{Fak}(2, \sigma : x/\text{barbier}) &: (\text{rasiert}(\text{barbier}, \text{barbier}))_4 \\ \text{Res}(3, 4) &: ()_5 \end{aligned}$$

und es gilt allgemein

Satz 3.6

Die Resolutionsregel (3.6) zusammen mit der Faktorisierungsregel (3.8) ist widerlegungsvollständig. Das heißt, durch Anwendung von Faktorisierungs- und Resolutionschritten lässt sich aus jeder unerfüllbaren Formel in konjunktiver Normalform die leere Klausel ableiten.

3.5.1 Resolutionsstrategien

So wichtig die Vollständigkeit der Resolution für den Anwender ist, so frustrierend kann in der Praxis die Suche nach einem Beweis sein. Der Grund ist der immens große kombinatorische Suchraum. Auch wenn am Anfang in $WB \wedge \neg Q$ nur wenige Paare von Klauseln mit unifizierbaren komplementären Literalen existieren, so erzeugt der Beweiser mit jedem Resolutionsschritt eine neue Klausel, welche die Zahl der im nächsten Schritt möglichen Resolutionsschritte vergrößert. Daher wird schon seit langem versucht, den Suchraum durch spezielle Strategien einzuschränken, möglichst ohne jedoch die Vollständigkeit zu verlieren. Die wichtigsten Strategien sind:

Die **Unit-Resolution** bevorzugt Resolutionsschritte, bei denen eine der beiden Klauseln aus nur einem Literal, einer so genannten Unit-Klausel, besteht. Diese Strategie erhält die Vollständigkeit und führt in vielen Fällen, aber nicht immer, zu einer Reduktion des Suchraumes. Sie gehört daher zu den heuristischen Verfahren (siehe Abschn. 6.3).

Eine garantierte Reduktion des Suchraumes erhält man durch die Anwendung der **Set of Support-Strategie**. Hier wird eine Teilmenge von $WB \wedge \neg Q$ als Set of Support (SOS) definiert. An jedem Resolutionsschritt muss eine Klausel aus dem SOS beteiligt sein und die Resolvente wird zum SOS hinzugefügt. Diese Strategie ist unvollständig. Sie wird vollständig, wenn sichergestellt ist, dass die Menge der Klauseln ohne das SOS erfüllbar ist (siehe Aufgabe 3.7). Oft wird als initiales SOS die negierte Anfrage $\neg Q$ verwendet.

Bei der **Input-Resolution** muss an jedem Resolutionsschritt eine Klausel aus der Eingabemenge $WB \wedge \neg Q$ beteiligt sein. Auch diese Strategie reduziert den Suchraum, aber auf Kosten der Vollständigkeit.

Bei der **Pure Literal-Regel** können alle Klauseln gelöscht werden, die Literale enthalten, zu denen es kein komplementäres Literal in anderen Klauseln gibt. Diese Regel reduziert den Suchraum und ist vollständig und wird daher von praktisch allen Resolutionsbeweisen verwendet.

Stellen die Literale einer Klausel K_1 eine Teilmenge der Literale der Klausel K_2 dar, so kann Klausel K_2 gelöscht werden. Zum Beispiel ist die Klausel

$$(regnet(\text{heute}) \Rightarrow \text{Straße_nass}(\text{heute}))$$

überflüssig, wenn schon $\text{Straße_nass}(\text{heute})$ gilt. Dieser wichtige Reduktionsschritt wird **Subsumption** genannt. Auch die Subsumption ist vollständig.

3.5.2 Gleichheit

Eine besonders unangenehme Quelle für die explosionsartige Vergrößerung des Suchraumes ist die Gleichheit. Fügt man die in (3.1) und (3.2) formulierten Gleichheitsaxiome zur Wissensbasis hinzu, so kann zum Beispiel die Symmetrieklausel $\neg x = y \vee y = x$ mit jeder positiven oder negierten Gleichung unifiziert werden, was zur Ableitung neuer Klauseln mit Gleichungen führt, auf die wieder Gleichheitsaxiome anwendbar sind, und so weiter. Ähnliche Auswirkungen haben auch die Transitivität und die Substitutionsaxiome. Daher wurden spezielle Inferenzregeln für die Gleichheit entwickelt, die ohne explizite Gleichheitsaxiome auskommen und insbesondere den Suchraum reduzieren. Die **Demodulation** zum Beispiel erlaubt die Ersetzung eines Terms t_1 durch t_2 , wenn eine Gleichung $t_1 = t_2$ existiert. Eine Gleichung $t_1 = t_2$ wird mittels Unifikation auf einen Term t wie folgt angewendet:

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

Etwas allgemeiner ist die **Paramodulation**, welche mit bedingten Gleichungen arbeitet [Bib92, BB92].

Eine Gleichung $t_1 = t_2$ erlaubt die Ersetzung des Terms t_1 durch t_2 genauso wie die Ersetzung von t_2 durch t_1 . Meist ist es sinnlos, eine durchgeführte Ersetzung wieder rückgängig zu machen. Vielmehr werden Gleichungen häufig genutzt um Terme zu vereinfachen. Sie werden also oft nur in einer Richtung genutzt. Gleichungen, die nur in einer Richtung genutzt werden, heißen gerichtete Gleichungen. Die effiziente Abarbeitung von gerichteten Gleichungen erfolgt mit so genannten **Termersetzungssystemen** (engl. term rewriting). Für Formeln mit vielen Gleichungen gibt es spezielle Gleichheitsbeweiser.

3.6 Automatische Theorembeweiser

Implementierungen von Beweiskalkülen auf Rechnern werden als Theorembeweiser bezeichnet. Neben Spezialbeweisern für Teilmengen von PL1 oder spezielle Anwendungen existieren heute eine ganze Reihe von automatischen Beweisern für die volle Prädikatenlogik und Logiken höherer Stufe, von denen hier nur einige wenige Systeme erwähnt werden sollen. Eine Übersicht über die wichtigsten Systeme ist zu finden bei [McC].

Einer der ältesten Resolutionsbeweiser wurde am Argonne National Laboratory in Chicago entwickelt. Basierend auf ersten Entwicklungen ab 1963 entstand im Jahr 1984 Otter [Kal01], der vor allem in Spezialgebieten der Mathematik erfolgreich angewendet wird, wie man der Homepage entnehmen kann:

Currently, the main application of Otter is research in abstract algebra and formal logic. Otter and its predecessors have been used to answer many open questions in the areas of finite semigroups, ternary Boolean algebra, logic calculi, combinatory logic, group theory, lattice theory, and algebraic geometry.

Einige Jahre später entstand an der Technischen Universität München basierend auf der schnellen Prolog-Technologie der Hochleistungsbeweiser SETHEO [LSBB92]. Mit dem Ziel, noch höhere Leistungen zu erreichen, wurde unter dem Namen PARTHEO eine Implementierung auf Parallelrechnern entwickelt. Es hat sich jedoch gezeigt, dass sich der Einsatz von Spezial-Hardware im Theorembeweisen, wie auch in anderen Bereichen der KI, nicht lohnt, denn diese Rechner werden sehr schnell von neuen, schnelleren Prozessoren und intelligenteren Algorithmen überholt. Auch aus München stammt E [Sch02], ein mit mehreren Preisen ausgezeichneter moderner Gleichheitsbeweiser, den wir im nächsten Beispiel kennenlernen werden. Auf der Homepage von E ist folgende kompakte, ironische Charakterisierung zu lesen, deren zweiter Teil übrigens für alle heute existierenden automatischen Beweiser zutrifft:

E is a purely equational theorem prover for clausal logic. That means it is a program that you can stuff a mathematical specification (in clausal logic with equality) and a hypothesis into, and which will then run forever, using up all of your machines resources. Very occasionally it will find a proof for the hypothesis and tell you so ;-).

Das Finden von Beweisen für wahre Aussagen ist offenbar so schwierig, dass die Suche nur äußerst selten, beziehungsweise nach sehr langer Zeit – wenn überhaupt – erfolgreich ist. Wir werden hierauf in Kap. 4 noch näher eingehen. An dieser Stelle sollte aber auch erwähnt werden, dass nicht nur Computer, sondern auch die meisten Menschen mit dem Finden von strengen formalen Beweisen ihre Mühe haben.

Wenn also offenbar die Computer alleine in vielen Fällen überfordert sind mit dem Beweisen, so liegt es nahe, Systeme zu bauen, die halbautomatisch arbeiten und eine enge Zusammenarbeit mit dem Nutzer erlauben. Dadurch kann der Mensch sein Wissen über die spezielle Anwendungsdomäne besser einbringen und so vielleicht die Suche nach dem Beweis stark einschränken. Einer der erfolgreichsten interaktiven Beweiser für Prädikatenlogik höherer Stufe ist Isabelle [NPW02], ein Gemeinschaftsprodukt der Universität Cambridge und der Technischen Universität München.

Wer einen leistungsfähigen Beweiser sucht, der sollte sich die aktuellen Ergebnisse der CASC (CADE ATP System Competition) ansehen [SS06].¹ Hier findet man als Sieger der Jahre 2001 bis 2006 in den Kategorien PL1 und Klauselnormalf orm den Beweiser Vampire aus Manchester, der mit einer Resolutionsvariante und spezieller Gleichheitsbehandlung arbeitet. Beim Beweisen von Gleichungen ist das System Waldmeister vom Max-Planck-Institut für Informatik in Saarbrücken seit Jahren führend.

Die vielen Spitzenplätze deutscher Systeme auf der CASC zeigen, dass deutsche Forschergruppen im Bereich des Automatischen Beweisens heute wie auch in der Vergangenheit eine führende Rolle spielen.

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

3.7 Mathematische Beispiele

Mit dem oben erwähnten Beweiser E [Sch02] wollen wir nun die Anwendung eines automatischen Beweisers demonstrieren. E ist ein spezialisierter Gleichheitsbeweiser, der durch eine optimierte Behandlung der Gleichheit den Suchraum stark verkleinert.

Wir wollen beweisen, dass in einer Halbgruppe links- und rechtsneutrales Element gleich sind. Zuerst formalisieren wir die Behauptung schrittweise.

Definition 3.10

Eine Struktur (M, \cdot) bestehend aus einer Menge M mit einer zweistelligen inneren Verknüpfung „ \cdot “ heißt Halbgruppe, wenn das Assoziativgesetz

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

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

Zu zeigen ist nun

Satz 3.7

Besitzt eine Halbgruppe ein links neutrales Element e_l und ein rechts neutrales Element e_r , so gilt $e_l = e_r$.

Zuerst beweisen wir den Satz halbformal per intuitivem mathematischem Schließen. Offenbar gilt für alle $x \in M$

$$e_l \cdot x = x \tag{3.9}$$

und

$$x \cdot e_r = x. \tag{3.10}$$

Setzen wir in (3.9) $x = e_r$ und in (3.10) $x = e_l$, so erhalten wir die beiden Gleichungen $e_l \cdot e_r = e_r$ und $e_l \cdot e_r = e_l$. Diese beiden Gleichungen zusammengefaßt ergeben

$$e_l = e_l \cdot e_r = e_r,$$

was zu beweisen war. Im letzten Schritt haben wir übrigens Symmetrie und Transitivität der Gleichheit angewendet.

Bevor wir den automatischen Beweiser anwenden, führen wir den Resolutionsbeweis manuell durch. Zuerst formalisieren wir die negierte Anfrage und die Wissensbasis WB , bestehend aus den Axiomen als Klauseln in konjunktiver Normalform

$$\begin{array}{ll} (\neg e_l = e_r)_1 & \text{negierte Anfrage} \\ (m(m(x, y), z) = m(x, m(y, z)))_2 & \\ (m(e_l, x) = x)_3 & \\ (m(x, e_r) = x)_4 & \end{array}$$

Gleichheitsaxiome:

$$\begin{array}{ll} (x = x)_5 & (\text{Reflexivität}) \\ (\neg x = y \vee y = x)_6 & (\text{Symmetrie}) \\ (\neg x = y \vee \neg y = z \vee x = z)_7 & (\text{Transitivität}) \\ (\neg x = y \vee m(x, z) = m(y, z))_8 & \text{Substitution in } m \\ (\neg x = y \vee m(z, x) = m(z, y))_9 & \text{Substitution in } m, \end{array}$$

wobei die Multiplikation durch das zweistellige Funktionssymbol m repräsentiert wird. Die Gleichheitsaxiome wurden analog zu (3.1) und (3.2) formuliert. Ein einfacher Resolutionsbeweis hat die Gestalt

$$\begin{aligned} \text{Res}(3, 6, x_6/m(e_l, x_3), y_6/x_3) : & \quad (x = m(e_l, x))_{10} \\ \text{Res}(7, 10, x_7/x_{10}, y_7/m(e_l, x_{10})) : & \quad (\neg m(e_l, x) = z \vee x = z)_{11} \\ \text{Res}(4, 11, x_4/e_l, x_{11}/e_r, z_{11}/e_l) : & \quad (e_r = e_l)_{12} \\ \text{Res}(1, 12, \emptyset) : & \quad () . \end{aligned}$$

Hierbei bedeutet zum Beispiel $\text{Res}(3, 6, x_6/m(e_l, x_3), y_6/x_3)$, dass beim Resolutionsschritt von Klausel 3 mit Klausel 6 die Variable x aus Klausel 6 durch $m(e_l, x_3)$ mit Variable x aus Klausel 3 substituiert wird. Analog wird y aus Klausel 6 durch x aus Klausel 3 ersetzt.

Nun wollen wir den Beweiser E auf das Problem anwenden. Die Klauseln werden transformiert in die Klauselnormalformsprache LOP durch die Abbildung

$$(\neg A_1 \vee \dots \vee \neg A_m \vee B_1 \vee \dots \vee B_n) \quad \mapsto \quad B_1 ; \dots ; B_n \leftarrow A_1 , \dots , A_m .$$

Die Syntax von LOP stellt eine Erweiterung der Prolog-Syntax (siehe Kap. 5) für Nicht-Hornklauseln dar. Damit erhalten wir als Eingabedatei für E

```

<- gl(el,er).
gl( m(m(X,Y),Z) , m(X,m(Y,Z)) ) .          # Query
gl( m(el,X) , X ) .                            # Assoziativität v. m
gl( m(X,er) , X ) .                            # linksneutrales El. v. m
gl(X,X) .                                     # rechtsneutrales El. v. m
gl(Y,X) <- gl(X,Y) .                          # Gleichhh: Reflexivität
gl(X,Z) <- gl(X,Y) , gl(Y,Z) .                # Gleichhh: Symmetrie
gl( m(X,Z) , m(Y,Z) ) <- gl(X,Y) .            # Gleichhh: Transitivität
gl( m(Z,X) , m(Z,Y) ) <- gl(X,Y) .            # Gleichhh: Substitution in m
gl( m(Z,X) , m(Z,Y) ) <- gl(X,Y) .            # Gleichhh: Substitution in m

```

wobei die Gleichheit durch das Prädikatssymbol `g1` modelliert wird. Der Aufruf des Beweisers liefert die folgende Ausgabe:

```
unixprompt> eproof halbgr1.lop
# Problem status determined, constructing proof object
# Evidence for problem status starts
  0 : [--g1(el,er)] : initial
  1 : [++g1(X1,X2),--g1(X2,X1)] : initial
  2 : [++g1(m(el,X1),X1)] : initial
  3 : [++g1(m(X1,er),X1)] : initial
  4 : [++g1(X1,X2),--g1(X1,X3),--g1(X3,X2)] : initial
  5 : [++g1(X1,m(X1,er))] : pm(3,1)
  6 : [++g1(X2,X1),--g1(X2,m(el,X1))] : pm(2,4)
  7 : [++g1(el,er)] : pm(5,6)
  8 : [] : sr(7,0)
  9 : [] : 8 : {proof}
# Evidence for problem status ends
```

Positive Literale sind durch `++` und negative durch `--` gekennzeichnet. In den hinten mit `initial` markierten Zeilen 0 bis 4 sind die Klauseln der Eingabedatei nochmal aufgelistet. `pm(a, b)` steht für einen Resolutionsschritt zwischen Klausel a und Klausel b. Man erkennt, dass der von E gefundene Beweis sehr ähnlich zu dem manuell erstellten Beweis ist. Da wir die Gleichheit explizit durch das Prädikat `g1` modellierten, konnte die besondere Stärke von E bei der Gleichheitsbehandlung nicht zum Tragen kommen. Wir lassen nun also die Gleichheitsaxiome weg und erhalten als Eingabedatei für den Beweiser

```
<- el = er.                                % Query
m(m(X,Y),Z) = m(X,m(Y,Z)) .               % Assoziativität v. m
m(el,X) = X .                             % linksneutrales El. v. m
m(X,er) = X .                            % rechtsneutrales El. v. m
```

Auch der Beweis wird kompakter. An der folgenden Ausgabe des Beweisers erkennt man, dass der Beweis im Wesentlichen in der Anwendung eines einzigen Inferenzschrittes auf die beiden relevanten Klauseln 1 und 2 besteht.

```
unixprompt> eproof halbgr1a.lop
# Problem status determined, constructing proof object
# Evidence for problem status starts
  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(4)
  6 : [] : 5 : {proof}
# Evidence for problem status ends
```

Der Leser möge sich nun selbst ein Bild über die Fähigkeiten von E machen (Aufgabe 3.9).

3.8 Anwendungen

In der Mathematik kommen automatische Theorembeweiser für spezielle Aufgabenstellungen durchaus zum Einsatz. Zum Beispiel wurde das wichtige Vierfarbentheorem aus der Graphentheorie 1976 mit Hilfe eines Spezialbeweisers erstmals bewiesen. Allerdings spielen die automatischen Beweiser in der Mathematik insgesamt eine untergeordnete Rolle.

Für die Entwicklung von Expertensystemen in praktischen Anwendungen hingegen war die Prädikatenlogik in der Anfangszeit der KI von großer Bedeutung. Aufgrund der Probleme bei der Modellierung von Unsicherheit (siehe Abschn. 4.4) werden Expertensysteme heute aber meist mit anderen Formalismen entwickelt (siehe hierzu auch [BKI00]).

Eine immer wichtigere Rolle spielt die Logik heute bei Verifikationsaufgaben. Die automatische Programmverifikation ist mittlerweile ein wichtiges Forschungsgebiet zwischen KI und Softwareentwicklung. Immer komplexere Softwaresysteme übernehmen heute immer mehr verantwortungsvolle und sicherheitsrelevante Aufgaben. Hier ist ein Beweis bestimmter Sicherheitseigenschaften des Programms wünschenswert. Durch Testen des fertigen Programms kann dieser Beweis nicht erbracht werden, denn es ist praktisch unmöglich, für alle möglichen Szenarien die Programme auf alle möglichen Eingaben anzuwenden. Dies ist also eine ideale Domäne für allgemeine oder auch spezialisierte Inferenzsysteme. Unter anderem sind heute kryptographische Protokolle im Einsatz, deren Sicherheitseigenschaften automatisch verifiziert wurden [FS97, Sch01]. Eine weitere Herausforderung für den Einsatz von automatischen Beweisern ist auch die Synthese von Soft- und Hardware. Hierbei sollen zum Beispiel Beweiser den Softwareentwickler bei der Generierung von Programmen aus Spezifikationen unterstützen [Kre06].

Auch die Wiederverwendung von Software ist heute von großer Bedeutung für viele Programmierer. Der Programmierer sucht nach einem Modul, das bei Eingabe von Daten mit bestimmten Eigenschaften ein Ergebnis berechnet mit den gewünschten Eigenschaften. Ein Sortieralgorithmus etwa akzeptiert als Eingabe Dateien mit Einträgen eines bestimmten Datentyps und erzeugt daraus eine Permutation dieser Einträge mit der Eigenschaft, dass der Nachfolger jedes Elements größer oder gleich dem Element ist. Der Programmierer formuliert nun als erstes in PL1 eine Spezifikation seiner Anfrage, die aus zwei Teilen besteht. Der erste Teil PRE_Q umfasst die Vorbedingungen, die vor der Anwendung des gesuchten Programms gelten müssen. Der zweite Teil $POST_Q$ enthält die Nachbedingungen, die nach der Anwendung des gesuchten Programms gelten müssen.

Im nächsten Schritt muss in einer Software-Datenbank nach Modulen gesucht werden, welche diese Anforderung erfüllen. Um dies formal zu checken, muss in der Datenbank zu jedem Modul M eine formale Beschreibung der Vorbedingungen PRE_M und der Nachbedingungen $POST_M$ gespeichert sein. Voraussetzung für die Tauglichkeit eines Moduls ist, dass zum einen die Vorbedingungen des Moduls aus den Vorbedingungen der Anfrage folgen. Es muss also gelten

$$PRE_Q \Rightarrow PRE_M.$$

Alle Bedingungen, die als Voraussetzung für die Anwendung des Moduls M gefordert werden, müssen auch als Vorbedingung in der Anfrage auftreten. Wenn zum Beispiel ein Modul in der Datenbank nur Listen von ganzen Zahlen akzeptiert, so müssen in der Vorbedingung der Anfrage auch Listen von ganzen Zahlen als Eingabe auftreten. Eine Zusatzbedingung in der Anfrage, dass zum Beispiel nur gerade Zahlen auftreten, stört hier nicht.

Außerdem muss für die Nachbedingungen

$$POST_M \Rightarrow POST_Q$$

gelten. Das heißt, nach der Anwendung des Moduls müssen alle Eigenschaften, die die Anfrage fordert, erfüllt sein. Wir zeigen nun die Anwendung eines Theorembeweisers auf diese Aufgabe an einem Beispiel aus [Sch01].

Beispiel 3.8

Als Sprache für die Spezifikation der Vor- und Nachbedingungen wird oft VDML-SL, die Vienna Development Method Specification Language verwendet. In der Software-Datenbank sei die Beschreibung eines Moduls ROTATE vorhanden, welches das erste Listenelement nach hinten an die letzte Stelle verschiebt. Gesucht ist ein Modul SHUFFLE, das eine beliebige Permutation (Vertauschung der Elemente) der Liste erzeugt. Die beiden Spezifikationen lauten:

$$\begin{array}{l} \text{ROTATE}(l : List) \ l' : List \\ \text{pre } true \\ \text{post} \\ \quad (l = [] \Rightarrow l' = []) \wedge \\ \quad (l \neq [] \Rightarrow l' = (\text{tail } l)^{\wedge}[\text{head } l]) \end{array}$$

$$\begin{array}{l} \text{SHUFFLE}(x : List) \ x' : List \\ \text{pre } true \\ \text{post } \forall i : Item . \\ \quad (\exists x_1, x_2 : List \cdot x = x_1^{\wedge}[i]^{\wedge}x_2 \Leftrightarrow \\ \quad \exists y_1, y_2 : List \cdot x' = y_1^{\wedge}[i]^{\wedge}y_2) \end{array}$$

Hierbei steht *true* für w (wahr), „ \wedge “ für die Konkatenation von Listen und „ \Leftrightarrow “ trennt Quantoren mit ihren Variablen vom Rest der Formel. Die Funktionen „*head* l “ und „*tail* l “ wählen aus einer Liste l ihr erstes Element, beziehungsweise den Rest der Liste aus. Die Spezifikation von SHUFFLE besagt, dass jedes Listenelement i , das vor der Anwendung von SHUFFLE in der Liste (x) war, nach der Anwendung im Ergebnis (x') sein muss, und umgekehrt. Zu beweisen ist nun, dass die Formel $(PRE_Q \Rightarrow PRE_M) \wedge (POST_M \Rightarrow POST_Q)$ aus der Wissensbasis mit einer Beschreibung des Datentyps *List* folgt. Aus den beiden VDML-SL-Spezifikationen ergibt sich also als Beweisaufgabe

$$\begin{aligned} & \forall l, l', x, x' : List \cdot (l = x \wedge l' = x' \wedge (w \Rightarrow w)) \wedge \\ & (l = x \wedge l' = x' \wedge ((l = [] \Rightarrow l' = []) \wedge (l \neq [] \Rightarrow l' = (\text{tail } l)^{\wedge}[\text{head } l])) \\ & \Rightarrow \forall i : Item \cdot (\exists x_1, x_2 : List \cdot x = x_1^{\wedge}[i]^{\wedge}x_2 \Leftrightarrow \exists y_1, y_2 : List \cdot x' = y_1^{\wedge}[i]^{\wedge}y_2)). \end{aligned}$$

Eine wichtige Anwendung der PL1 in den nächsten Jahren wird voraussichtlich das **Semantic Web** darstellen. Die Inhalte des World Wide Web sollen nicht nur für Menschen, sondern auch für Maschinen interpretierbar werden. Dazu werden die Web-Seiten zusätzlich mit einer Beschreibung ihrer Semantik in einer formalen Beschreibungssprache versehen. Dadurch wird zum Beispiel das Suchen nach Informationen im Web wesentlich effektiver als heute, wo im Wesentlichen nach syntaktischen Textbausteinen gesucht werden kann.

Als Beschreibungssprachen werden entscheidbare Teilmengen der Prädikatenlogik verwendet. Ganz wichtig und eng verbunden mit den Beschreibungssprachen ist die Entwicklung von effizienten Kalkülen für das Schließen. Eine Anfrage für eine zukünftige semantisch arbeitende Suchmaschine könnte (informal) zum Beispiel lauten: An welchen Orten in der Schweiz sind am kommenden Sonntag in Regionen unterhalb 2000m die Skipisten optimal präpariert und das Wetter gut? Zur Beantwortung solch einer Anfrage werden Kalküle benötigt, die sehr schnell auf riesigen Mengen von Fakten und Regeln arbeiten können. Weniger wichtig sind hier komplexe geschachtelte Funktionsterme.

Als grundlegende Beschreibungssprache wurde vom World Wide Web Consortium die Sprache RDF (Resource Description Framework) entwickelt. Aufbauend auf RDF erlaubt die wesentlich mächtigere Sprache OWL (Web Ontology Language) die Beschreibung von Relationen zwischen Objekten und Klassen von Objekten, ähnlich wie in PL1 [PB06]. **Ontologien** sind Beschreibungen von Relationen zwischen möglichen Objekten.

Eine Schwierigkeit beim Aufbau der Beschreibungen für die unzähligen Webseiten wird der Arbeitsaufwand und auch die Überprüfung der Korrektheit der semantischen Beschreibungen sein. Sehr hilfreich könnten hier maschinelle Lernverfahren zur automatischen Generierung der Beschreibungen sein. Ein interessanter Ansatz zur „automatischen“ Generierung von Semantik im Web wurde vorgestellt von Luis von Ahn von der Carnegie Mellon University [vA06]. Er hat Computerspiele entwickelt, bei denen Spieler gemeinsam verteilt Bilder mit Begriffen beschreiben sollen. So wird auf spielerische Weise und ohne Kosten den Bildern im Web Semantik zugeordnet. In Aufgabe 3.10 möge der Leser diese Spiele testen und sich den Vortrag des Erfinders anhören.

3.9 Zusammenfassung

Die wichtigsten Grundlagen, Begriffe und Verfahren der Prädikatenlogik stehen nun bereit, und wir haben gezeigt, dass sogar eine der schwierigsten intellektuellen Aufgaben, nämlich das Beweisen mathematischer Sätze, automatisierbar ist. Automatische Beweiser sind nicht nur in der Mathematik, sondern vor allem für Verifikationsaufgaben in der Informatik einsetzbar. Beim Schließen im Alltag ist die Prädikatenlogik allerdings meist nicht geeignet. Im nächsten und den folgenden Kapiteln werden wir die Schwachstellen und interessante moderne Alternativen aufzeigen. Außerdem werden wir in Kap. 5 zeigen, dass man mit Logik und prozeduralen Erweiterungen auch elegant programmieren kann.

Eine sehr gute moderne Einführung in die Logik mit kompakter Darstellung von Erweiterungen wie Modallogik und Temporallogik ist zu finden in [Das05]. Wer sich für die Details der Resolution und anderer Kalküle für automatische Beweiser interessiert, der findet in [BB92], [Bib92], [GRS03] und [CL73] gute weiterführende Lehrbücher. Verweise auf Ressourcen im Internet sind zu finden auf der Webseite zum Buch.

3.10 Übungen

Aufgabe 3.1 Gegeben sei das dreistellige Prädikat „kind“ und das einstellige Prädikat „weiblich“ aus Beispiel 3.2. Definieren Sie ein

- einstelliges Prädikat „männlich“.
- zweistellige Prädikate „vater“ und „mutter“.
- zweistelliges Prädikat „geschwister“.
- Prädikat „eltern(x, y, z)“, das genau dann wahr ist, wenn x Vater und y Mutter von z sind.
- Prädikat „onkel(x, y)“, das genau dann wahr ist, wenn x Onkel von y ist (verwenden Sie dazu die schon definierten Prädikate).
- zweistelliges Prädikat „vorfahr“ mit der Bedeutung: *Vorfahren sind Eltern, Großeltern, etc. in beliebig vielen Generationen.*

Aufgabe 3.2 Formalisieren Sie folgende Aussagen in Prädikatenlogik:

- Jeder Mensch hat einen Vater und eine Mutter.
- Manche Menschen haben Kinder.
- Alle Vögel fliegen.
- Es gibt ein Tier das (manche) Körner fressende Tiere frisst.
- Jedes Tier frisst Pflanzen oder pflanzenfressende Tiere, die viel kleiner sind als es selbst.

Aufgabe 3.3 Bearbeiten Sie Aufgabe 3.1, indem Sie statt Prädikaten für „vater“ und „mutter“ einstellige Funktionssymbole und Gleichheit verwenden.

Aufgabe 3.4 Geben Sie prädikatenlogische Axiome für die zweistellige Relation „ $<$ “ als totale Ordnung an. Für eine totale Ordnung muss gelten: 1. Je zwei Elemente sind vergleichbar, 2. Sie ist antisymmetrisch, 3. Sie ist transitiv.

Aufgabe 3.5 Unifizieren Sie (falls möglich) folgende Terme und geben Sie den MGU und den resultierenden Term an.

- $p(x, f(y)), p(f(z), u)$
- $p(x, f(x)), p(y, y)$
- $x = 4 - 7 \cdot x, \cos y = z$
- $x < 2 \cdot x, 3 < 6$
- $q(f(x, y, z), f(g(w, w), g(x, x), g(y, y))), q(u, u)$

Aufgabe 3.6

- a) Transformieren Sie die Russellsche Antinomie aus Beispiel 3.7 in KNF.
- b) Zeigen Sie, dass sich mit Resolution ohne Faktorisierung aus (3.7) die leere Klausel nicht ableiten lässt. Versuchen Sie, dies intuitiv zu verstehen.

Aufgabe 3.7

- a) Warum ist Resolution mit der Set of Support-Strategie unvollständig?
- b) Begründen Sie (ohne Beweis), weshalb die Set of Support-Strategie vollständig wird, wenn $(WB \wedge \neg Q) \setminus SOS$ erfüllbar ist.
- c) Warum ist Resolution mit der Pure Literal-Regel vollständig?

Aufgabe 3.8 * Formalisieren und beweisen Sie mit Resolution, dass in einer Halbgruppe mit mindestens zwei verschiedenen Elementen a, b sowie einem linksneutralen Element e und einem Links-Nullelement n diese beiden Elemente verschieden sein müssen, d. h. es gilt $n \neq e$. Verwenden Sie Demodulation, das heißt, Sie dürfen in den Regeln „Gleiches durch Gleiches“ ersetzen.

Aufgabe 3.9 Besorgen Sie sich den Theorembeweiser E [Sch02] oder einen anderen Beweiser und beweisen Sie folgende Aussagen. Vergleichen Sie die Beweise mit denen im Skript.

- a) Die Behauptung aus Beispiel 2.3.
- b) Die Russellsche Antinomie aus Beispiel 3.7.
- c) Die Behauptung aus Aufgabe 3.8.

Aufgabe 3.10 Testen Sie die Spiele ESP-game und Squigl auf www.gwap.com, welche dazu dienen, Bildern im Web Semantik zuzuordnen. Hören Sie sich dann den Vortrag über Human Computation von Luis von Ahn an: <http://video.google.de/videoplay?docid=8246463980976635143>

4.1 Das Suchraumproblem

Wie schon an verschiedenen Stellen erwähnt, gibt es auf der Suche nach einem Beweis fast immer in jedem Schritt viele (je nach Kalkül eventuell sogar unendlich viele) Möglichkeiten für die Anwendung von Inferenzregeln. Es ergibt sich dadurch das schon erwähnte explosionsartige Wachsen des Suchraums (Abb. 4.1). In Worst-Case müssen zum Finden eines Beweises alle diese Möglichkeiten versucht werden, was in für Menschen zumutbaren Zeiträumen aber meist nicht möglich ist.

Vergleicht man nun automatische Beweiser oder Inferenzsysteme mit Mathematikern oder menschlichen Experten, die Erfahrung im Schließen in speziellen Domänen haben, so macht man interessante Beobachtungen. Zum einen können erfahrene Mathematiker Sätze beweisen, die für alle automatischen Beweiser weit außerhalb deren Reichweite liegen. Andererseits jedoch schaffen automatische Beweise zigtausende Inferenzen pro Sekunde. Ein Mensch hingegen schafft vielleicht eine Inferenz pro Sekunde. Obwohl menschliche Experten also auf der Objektebene (d.h. dem Ausführen der Inferenzen) viel langsamer sind, lösen sie die schwierigen Probleme offenbar viel schneller.

Hierfür gibt es mehrere Gründe. Wir Menschen benutzen intuitive Kalküle, die auf höherer Ebene arbeiten und oft viele der einfachen Inferenzen eines Beweisers in einem Schritt ausführen. Außerdem arbeiten wir mit Lemmas, das heißt abgeleiteten wahren Formeln, die wir schon kennen und damit nicht jedes Mal neu beweisen müssen. Es gibt mittlerweile auch maschinelle Beweiser, die mit derartigen Verfahren arbeiten. Aber auch sie können mit menschlichen Experten noch nicht konkurrieren.

Ein weiterer, ganz wichtiger Vorteil von uns Menschen ist die Intuition und die Anschauung, ohne die wir keine schwierigen Probleme lösen könnten [Pól95]. Beim Versuch, die Intuition zu formalisieren, treten Probleme auf. Die Erfahrung in angewandten KI-Projekten zeigt, dass in komplexen Domänen wie zum Beispiel 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. Also kann

Abb. 4.1 Mögliche Folgen der Explosion eines Suchraums ...



man dieses Wissen auch nicht programmieren oder in Kalküle in Form von **Heuristiken** integrieren. Heuristiken sind Verfahren, die in vielen Fällen den Weg zum Ziel stark vereinfachen oder verkürzen, die aber auch (eher selten) den Weg zum Ziel stark verlängern können. Heuristische Suche ist nicht nur in der Logik, sondern generell beim Problemlösen von großer Bedeutung in der KI und wird daher ausführlich in Kap. 6 behandelt.

Ein interessanter Ansatz, der seit etwa 1990 verfolgt wird, ist die Anwendung von maschinellen Lernverfahren zum Lernen von Heuristiken für die Steuerung der Suche von Inferenzsystemen, worauf wir kurz eingehen wollen. Ein Resolutionsbeweiser etwa hat während der Suche nach einem Beweis in jedem Schritt eventuell viele hunderte oder mehr Möglichkeiten für Resolutionsschritte, aber nur einige wenige führen zum Ziel. Ideal wäre es, wenn der Beweiser ein Orakel fragen könnte, welche zwei Klauseln er im nächsten Schritt verwenden soll, um schnell den Beweis zu finden. Man versucht nun, solche Beweisteuerungsmodelle zu bauen, die die verschiedenen Alternativen für den nächsten Schritt heuristisch bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Fall der Resolution könnte die Bewertung der verfügbaren Klauseln über eine Funktion erfolgen, die aufgrund von bestimmten Attributen der Klauseln wie etwa die Zahl der Literale, die Zahl der positiven Literale, die Komplexität der Terme, etc. für jedes Paar von resolvierbaren Klauseln einen Wert berechnet.

Wie wird diese Funktion nun implementiert? Da dieses Wissen „intuitiv“ ist, kennt es der Programmierer nicht. Man versucht, die Natur zu kopieren und verwendet stattdessen maschinelle Lernverfahren, um aus erfolgreichen Beweisen zu lernen [ESS89, SE90]. Die Attribute aller an erfolgreichen Resolutionsschritten beteiligten Klauselpaare werden als positiv gespeichert, und die Attribute aller erfolglosen Resolutionen werden als negativ ge-

speichert. Dann wird mit einem maschinellen Lernverfahren aus diesen Trainingsdaten ein Programm generiert, welches Klauselpaare heuristisch bewerten kann (siehe Abschn. 9.5).

Ein anderer erfolgreicher Ansatz, das mathematische Schließen zu verbessern, wird verfolgt bei interaktiven Systemen, die unter Kontrolle des Benutzers arbeiten. Hier sind zum Beispiel Computeralgebraprogramme wie Mathematica, Maple oder MuPad zu nennen, die schwierige symbolische mathematische Manipulationen automatisch ausführen können. Die Suche nach dem Beweis bleibt aber vollständig dem Menschen überlassen. Deutlich mehr Unterstützung bei der Beweissuche bietet zum Beispiel der in Abschn. 3.6 schon erwähnte interaktive Beweiser Isabelle [NPW02]. Es gibt derzeit mehrere Projekte wie zum Beispiel Omega [SB04] oder MKM¹ zur Entwicklung von Systemen für die Unterstützung von Mathematikern beim Beweisen.

Zusammenfassend kann man sagen, dass aufgrund der Suchraumproblematik automatische Beweiser heute meist nur relativ einfache Sätze in speziellen Domänen mit wenigen Axiomen beweisen können.

4.2 Entscheidbarkeit und Unvollständigkeit

Die Prädikatenlogik erster Stufe bietet ein mächtiges Werkzeug zur Repräsentation von Wissen und zum Schließen. Wir wissen, dass es korrekte und vollständige Kalküle und Theorembeweiser gibt. Beim Beweis eines Satzes, das heißt einer wahren Aussage, ist solch ein Beweiser sehr hilfreich, denn nach endlicher Zeit weiß man aufgrund der Vollständigkeit, dass die Aussage wirklich wahr ist. Was aber, wenn die Aussage nicht wahr ist? Darüber macht der Vollständigkeitssatz (Satz 3.3) keine Aussage.² Es gibt nämlich kein Verfahren, das jede Formel aus PL1 nach endlicher Zeit beweist oder widerlegt, denn es gilt

Satz 4.1

Die Menge der allgemeingültigen Formeln der Prädikatenlogik erster Stufe ist halbentscheidbar.

Dieser Satz besagt, dass es Programme (Theorembeweiser) gibt, die bei Eingabe einer wahren (allgemeingültigen) Formel nach endlicher Zeit die Wahrheit feststellen. Ist eine Formel aber nicht allgemeingültig, so kann es sein, dass der Beweiser nicht hält. (Der Leser möge sich in Aufgabe 4.1 mit dieser Frage auseinandersetzen.) Die Aussagenlogik ist entscheidbar, denn die Wahrheitstafelmethode liefert in endlicher Zeit alle Modelle einer Formel. Offenbar ist die Prädikatenlogik mit den Quantoren und geschachtelten Funktionsymbolen eine etwas zu mächtige Sprache, um noch entscheidbar zu sein.

¹ <http://www.mathweb.org/mathweb/demo.html>

² Gerade dieser Fall ist für die Praxis besonders wichtig, denn wenn ich schon weiß, dass eine Aussage wahr ist, brauche ich keinen Beweiser mehr.

Andererseits ist die Prädikatenlogik für viele Zwecke noch nicht mächtig genug. Oft möchte man Aussagen über Mengen von Prädikaten oder Funktionen machen. Dies geht mit PL1 nicht, denn sie kennt nur Quantoren für Variablen, nicht aber für Prädikate oder Funktionen.

Kurt Gödel hat kurz nach seinem Vollständigkeitssatz für PL1 gezeigt, dass die Vollständigkeit verloren geht, wenn man PL1 auch nur minimal erweitert, um eine Logik höherer Stufe zu bauen. Eine Logik erster Stufe kann nur über Variablen quantifizieren. Eine Logik zweiter Stufe kann auch über Formeln der ersten Stufe quantifizieren, und eine Logik dritter Stufe kann über Formeln der zweiten Stufe quantifizieren. Sogar, wenn man nur das Induktionsaxiom für die natürlichen Zahlen dazunimmt, wird die Logik schon unvollständig. Die Aussage „*Wenn ein Prädikat $p(n)$ für n gilt, so gilt auch $p(n+1)$* “, beziehungsweise

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

ist eine Aussage zweiter Stufe, denn es wird über ein Prädikat quantifiziert. Gödel hat folgenden Satz bewiesen:

Satz 4.2 (Gödelscher Unvollständigkeitssatz)

Jedes Axiomensystem für die Natürlichen Zahlen mit Addition und Multiplikation (die Arithmetik) ist unvollständig. Das heißt, es gibt in der Arithmetik wahre Aussagen, die nicht beweisbar sind.

Der Gödelsche Beweis arbeitet mit der so genannten Gödelisierung. Hier wird jede arithmetische Formel als Zahl kodiert. Sie erhält eine eindeutige Gödelnummer. Die Gödelisierung wird nun verwendet um die Formel

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

in der Sprache der Arithmetik zu formulieren. Diese Formel ist wahr aus folgendem Grund. Angenommen, F ist falsch. Dann kann man F also beweisen und hat damit gezeigt, dass F nicht beweisbar ist. Dies ist ein Widerspruch. Also ist F wahr und damit nicht beweisbar.

Der tiefere Hintergrund dieses Satzes ist der, dass mathematische Theorien (Axiomensysteme) und allgemeiner Sprachen unvollständig werden, wenn die Sprache zu mächtig wird. Ein ähnliches Beispiel ist die Mengenlehre. Diese Sprache ist so mächtig, dass sich mit ihr Antinomien formulieren lassen. Dies sind Aussagen, die sich selbst widersprechen, wie die aus Beispiel 3.7 schon bekannte Aussage mit den Barbieren die alle rasieren, die sich nicht selbst rasieren (siehe Aufgabe 4.2).³ Das Dilemma besteht darin, dass wir uns mit Sprachen, die mächtig genug sind, um elegant die Mathematik und interessante

³ Viele weitere logische Antinomien sind zu finden in [Wie].

Anwendungen zu beschreiben, durch die Hintertür derartige Widersprüche oder Unvollständigkeiten einhandeln. Dies heißt jedoch nicht, dass Logiken höherer Stufe für formale Methoden völlig ungeeignet sind. Es gibt durchaus formale Systeme und auch Beweiser für Logiken höherer Stufe.

4.3 Der fliegende Pinguin

An einem ganz einfachen Beispiel wollen wir ein fundamentales Problem der Logik und mögliche Lösungsansätze aufzeigen. Gegeben seien die Aussagen

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

Formalisiert in PL1 ergibt sich als Wissensbasis WB

$$\begin{aligned} & pinguin(tweety) \\ & pinguin(x) \Rightarrow vogel(x) \\ & vogel(x) \Rightarrow fliegen(x) \end{aligned}$$

Daraus lässt sich (zum Beispiel mit Resolution) $fliegen(tweety)$ ableiten (siehe Abb. 4.2).⁴ Offenbar ist die Formalisierung der Flugeigenschaften von Pinguinen noch unzureichend. Versuchen wir es mit der zusätzlichen Aussage *Pinguine können nicht fliegen*, das heißt mit

$$pinguin(x) \Rightarrow \neg fliegen(x)$$

Daraus lässt sich nun $\neg fliegen(tweety)$ ableiten. Aber $fliegen(tweety)$ gilt immer noch. Die Wissensbasis ist also widersprüchlich. Wir erkennen hier eine wichtige Eigenschaft der Logik, nämlich die Monotonie. Obwohl wir explizit aussagen, dass Pinguine nicht fliegen können, lässt sich immer noch das Gegenteil ableiten.

Definition 4.1

Eine Logik heißt monoton, wenn für eine beliebige Wissensbasis WB und eine beliebige Formel ϕ die Menge der aus WB ableitbaren Formeln eine Teilmenge der aus $WB \cup \phi$ ableitbaren Formeln ist.

⁴ Die formale Durchführung dieses und der folgenden einfachen Beweise sei dem Leser zur Übung überlassen (Aufgabe 4.3).

Abb. 4.2 Der fliegende Pinguin Tweety



Wird eine Formelmenge erweitert, so können nach der Erweiterung also alle vorher ableitbaren Aussagen immer noch bewiesen werden, und eventuell weitere zusätzliche Aussagen. Die Menge der beweisbaren Aussagen wächst also monoton, wenn die Formelmenge erweitert wird. Für unser Beispiel heißt das, dass das Erweitern der Wissensbasis nie zum Ziel führen wird. Ändern wir also WB ab, indem wir die offensichtlich falsche Aussage „(alle) Vögel können fliegen“ durch die exaktere Aussage „(alle) Vögel außer Pinguinen können fliegen“ ersetzen und als WB_2 folgende Klauseln erhalten:

$$\begin{aligned} & \text{pinguin}(tweety) \\ & \text{pinguin}(x) \Rightarrow \text{vogel}(x) \\ & \text{vogel}(x) \wedge \neg \text{pinguin}(x) \Rightarrow \text{fliegen}(x) \\ & \text{pinguin}(x) \Rightarrow \neg \text{fliegen}(x) \end{aligned}$$

Nun ist die Welt scheinbar wieder in Ordnung. Wir können $\neg \text{fliegen}(tweety)$ ableiten, $\text{fliegen}(tweety)$ aber nicht, weil wir hierfür $\neg \text{pinguin}(x)$ benötigen würden, was aber nicht ableitbar ist. Solange es in dieser Welt nur Pinguine gibt, herrscht Friede. Jeder normale Vogel hingegen führt aber sofort zu Problemen. Wir wollen den Raben Abraxas hinzunehmen und erhalten WB_3 :

$$\begin{aligned} & \text{rabe}(abraxas) \\ & \text{rabe}(x) \Rightarrow \text{vogel}(x) \\ & \text{pinguin}(tweety) \\ & \text{pinguin}(x) \Rightarrow \text{vogel}(x) \\ & \text{vogel}(x) \wedge \neg \text{pinguin}(x) \Rightarrow \text{fliegen}(x) \\ & \text{pinguin}(x) \Rightarrow \neg \text{fliegen}(x) \end{aligned}$$

Über die Flugeigenschaften von Abraxas können wir hieraus gar nichts ableiten, denn wir haben vergessen zu formulieren, dass Raben keine Pinguine sind. Also erweitern wir WB_3

zu WB_4 :

```
rabe(abraxas)
rabe(x) ⇒ vogel(x)
rabe(x) ⇒ ¬pinguin(x)
pinguin(tweety)
pinguin(x) ⇒ vogel(x)
vogel(x) ∧ ¬pinguin(x) ⇒ fliegen(x)
pinguin(x) ⇒ ¬fliegen(x)
```

Die für uns Menschen selbstverständliche Tatsache, dass Raben keine Pinguine sind, muss hier explizit hinzugefügt werden. Für den Aufbau einer Wissensbasis mit allen etwa 9800 Vogelarten weltweit muss also für jede Vogelart (außer für Pinguine) angegeben werden, dass sie nicht zu den Pinguinen gehört. Analog muss man für alle anderen Ausnahmen wie zum Beispiel den Vogel Strauß auch vorgehen.

Für jedes Objekt in der Wissensbasis müssen neben seinen Eigenschaften auch alle Eigenschaften aufgeführt werden, die das Objekt nicht hat.

Zur Lösung dieses Problems wurden verschiedene Formen nichtmonotoner Logiken entwickelt, welche das Entfernen von Wissen (Formeln) aus der Wissensbasis ermöglichen. Unter dem Namen **Default-Logik** wurden Logiken entwickelt, die es erlauben, bestimmte Eigenschaften für Objekte zu vergeben, die so lange gelten wie keine anderen Regeln vorhanden sind. Im Tweety-Beispiel wäre die Regel *Vögel können fliegen* eine derartige **Default-Regel**. Trotz großer Anstrengungen haben sich diese Logiken aufgrund von semantischen und praktischen Problemen bis heute nicht durchgesetzt.

Besonders unangenehm kann die Monotonie bei komplexen Planungsproblemen werden, bei denen sich die Welt verändern kann. Wird zum Beispiel ein blaues Haus rot angestrichen, so ist es hinterher rot. Eine Wissensbasis wie zum Beispiel

```
farbe(haus,blau)
anstreichen(haus,rot)
anstreichen(x,y) ⇒ farbe(x,y)
```

führt zu dem Schluss, dass nach dem Anstreichen das Haus rot und blau ist. Das Problem, das sich hier beim Planen auftut, ist unter dem Namen **Frame-Problem** bekannt. Ein Lösung hierfür stellt der in Abschn. 5.6 vorgestellte Situationskalkül dar.

Ein in bestimmten Fällen sehr interessanter Ansatz zur Modellierung von Problemstellungen wie im Tweety-Beispiel ist die Wahrscheinlichkeitstheorie. Die Aussage „Alle Vögel können fliegen“ ist falsch. Korrekt ist eine Aussage wie etwa „Fast alle Vögel können fliegen“. Noch exakter wird diese Aussage, wenn man die Wahrscheinlichkeit für „Vögel können fliegen“ angibt. Dies führt zur **Wahrscheinlichkeitslogik** (engl. probabilistic logic), die heute ein wichtiges Teilgebiet der KI und ein wichtiges Werkzeug zur Modellierung von Unsicherheit darstellt (siehe Kap. 7).

4.4 Modellierung von Unsicherheit

Die zweiwertige Logik kann und soll nur Sachverhalte modellieren, bei denen es *wahr* und *falsch* und keine anderen Wahrheitswerte gibt. Für viele Aufgaben des Schließens im Alltag ist die zweiwertige Logik daher zu ausdrucksschwach. Die Regel

$$\text{vogel}(x) \Rightarrow \text{fliegen}(x)$$

ist wahr für fast alle Vögel, aber für manche ist sie falsch. Wie schon erwähnt, bietet es sich hier an, zur exakten Formulierung der Unsicherheit, mit Wahrscheinlichkeiten zu arbeiten. Die Aussage „99 % aller Vögel können fliegen“ lässt sich zum Beispiel durch den Ausdruck

$$P(\text{vogel}(x) \Rightarrow \text{fliegen}(x)) = 0,99$$

formalisieren. In Kap. 7 werden wir sehen, dass es besser ist, hier mit bedingten Wahrscheinlichkeiten wie zum Beispiel

$$P(\text{fliegen} | \text{vogel}) = 0,99$$

zu arbeiten. Mit Hilfe von **Bayes-Netzen** lassen sich auch komplexe Anwendungen mit vielen Variablen modellieren.

Eine andere Modellierung wird benötigt für die Aussage „Das Wetter ist schön“. Hier von wahr oder falsch zu reden macht oft wenig Sinn. Die Variable *Wetter_ist_schön* sollte nicht binär modelliert werden, sondern stetig mit Werten zum Beispiel im Intervall [0, 1]. *Wetter_ist_schön* = 0,7 bedeutet dann „Das Wetter ist ziemlich schön“. Für derartige stetige (unscharfe) Variablen wurde die **Fuzzy-Logik** entwickelt, die auch in Kap. 7 vorgestellt wird.

Die Wahrscheinlichkeitstheorie bietet zusätzlich die Möglichkeit, Aussagen über die Wahrscheinlichkeit von stetigen Variablen zu machen. Eine Aussage im Wetterbericht wie zum Beispiel „Mit hoher Wahrscheinlichkeit wird es ein bisschen Regen geben“ könnte zum Beispiel durch eine Wahrscheinlichkeitsdichte der Form

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

exakt formuliert und graphisch wie etwa in Abb. 4.3 dargestellt werden. Daraus kann man zum Beispiel ablesen, dass es sehr wahrscheinlich zwischen 10 und 20 mm Regen geben wird.

Diese sehr allgemeine und auch anschauliche Darstellung der beiden angesprochenen Arten von Unsicherheit zusammen mit der induktiven Statistik und der Theorie der Bayes-Netze ermöglicht die Beantwortung beliebiger probabilistischer Anfragen.

Sowohl die Wahrscheinlichkeitslogik als auch die Fuzzy-Logik sind nicht direkt mit der Prädikatenlogik vergleichbar, da sie keine Variablen und Quantoren erlauben. Sie sind also wie folgt als Erweiterungen der Aussagenlogik zu sehen:

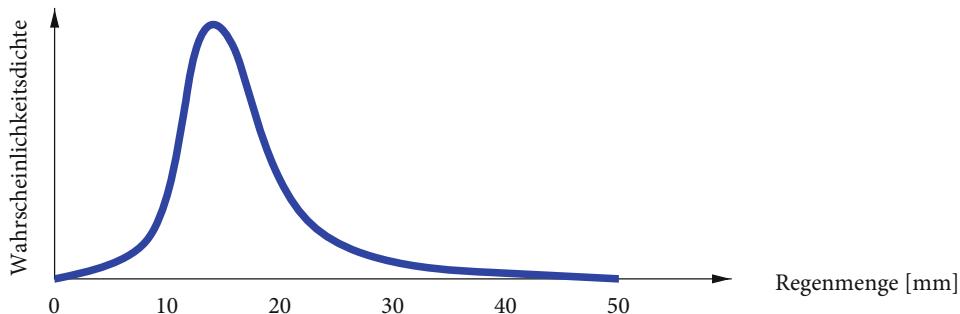


Abb. 4.3 Wahrscheinlichkeitsdichte der stetigen Variable Regenmenge

Formalismus	Anzahl der Wahrheitswerte	Wahrscheinlichkeiten ausdrückbar
Aussagenlogik	2	–
Fuzzy-Logik	∞	–
diskrete Wahrscheinlichkeitslogik	n	ja
stetige Wahrscheinlichkeitslogik	∞	ja

4.5 Übungen

Aufgabe 4.1 *

- a) Mit folgendem (falschen) Argument könnte jemand behaupten, PL1 sei entscheidbar:
Man nehme einen vollständigen Beweiskalkül für PL1. Für jede wahre Formel findet man damit in endlicher Zeit einen Beweis. Für alle anderen Formeln ϕ gehe ich wie folgt vor: Ich wende den Kalkül auf $\neg\phi$ an und zeige, dass $\neg\phi$ wahr ist. Also ist ϕ falsch. Damit kann ich jede Formel aus PL1 beweisen oder widerlegen. Finden Sie den Fehler in der Argumentation und ändern Sie diese so ab, dass sie korrekt wird.
- b) Konstruieren Sie ein Entscheidungsverfahren für die Menge der wahren und unerfüllbaren Formeln aus PL1.

Aufgabe 4.2

- a) Gegeben sei die Aussage „*Es gibt einen Barbier, der alle Menschen rasiert, die sich nicht selbst rasieren.*“ Überlegen Sie sich, ob sich dieser Barbier selbst rasiert.
- b) Es sei $M = \{x \mid x \notin x\}$. Beschreiben Sie diese Menge und überlegen Sie sich, ob M sich selbst enthält.

Aufgabe 4.3

Verwenden Sie einen automatischen Beweiser (zum Beispiel E [[Sch02](#)]) und wenden Sie ihn auf alle fünf verschiedenen Axiomatisierungen des Tweety-Beispiels aus Abschn. 4.3 an. Bestätigen Sie die dort gemachten Aussagen.

Im Vergleich zu klassischen Programmiersprachen wie zum Beispiel C oder Pascal bietet die Logik die Möglichkeit, Zusammenhänge elegant, kompakt und deklarativ zu beschreiben. Automatische Theorembeweiser sind sogar in der Lage, zu entscheiden, ob eine Anfrage logisch aus einer Wissensbasis folgt. Hierbei sind der Beweiskalkül und das in der Wissensbasis gespeicherte Wissen streng getrennt. Eine in Klauselnnormalform aufgeschriebene Formel kann als Eingabe für jeden Theorembeweiser verwendet werden, unabhängig vom verwendeten Beweiskalkül. Für die Repräsentation von Wissen und das Schließen ist dies von großem Nutzen.

Will man jedoch Algorithmen implementieren, die zwangsläufig (auch) prozedurale Komponenten haben, so genügt die rein deklarative Beschreibung oft nicht. Robert Kowalski, einer der Pioniere in der Logikprogrammierung, hat dies mit der Formel

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

auf den Punkt gebracht. Mit der Sprache Prolog hat sich diese Idee durchgesetzt. Vor allem in der KI und in der Computerlinguistik wird Prolog in vielen Projekten eingesetzt. Wir geben hier eine kurze Einführung in diese Sprache, stellen die wichtigsten Konzepte vor, zeigen die Stärken auf und vergleichen sie mit anderen Programmiersprachen und mit Theorembeweisern. Wer einen kompletten Programmierkurs sucht, wird verwiesen auf Lehrbücher wie [Bra86, CM94] und die Handbücher [Wie04, Dia04].

Die Syntax der Sprache Prolog kennt nur Hornklauseln. In folgender Tabelle sind die logische Schreibweise und die Prologsyntax nebeneinander gestellt.

PL1/Klauselnnormalform	Prolog	Bezeichnung
$(\neg A_1 \vee \dots \vee \neg A_m \vee B)$	$B : - A_1, \dots, A_m.$	Regel
$(A_1 \wedge \dots \wedge A_m) \Rightarrow B$	$B : - A_1, \dots, A_m.$	Regel
A	$A.$	Fakt
$(\neg A_1 \vee \dots \vee \neg A_m)$	$? - A_1, \dots, A_m.$	Anfrage (Query)
$\neg(A_1 \wedge \dots \wedge A_m)$	$? - A_1, \dots, A_m.$	Anfrage (Query)

Hierbei sind A_1, \dots, A_m, A, B Literale. Die Literale sind wie in PL1 aufgebaut aus einem Prädikatssymbol mit Termen als Argumente. Wie man schon an obiger Tabelle erkennt, gibt es eine Negation im strengen logischen Sinne in Prolog nicht, denn das Vorzeichen jedes Literals ist durch seine Position in der Klausel bestimmt.

5.1 Prolog-Systeme und Implementierungen

In der Linkssammlung auf der Homepage zum Buch ist eine Übersicht der aktuellen Prolog-Systeme verfügbar. Dem Leser empfehle ich die sehr leistungsfähigen und frei (GNU public licence) verfügbaren 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 auf der Basis der **Warren-abstract-machine (WAM)**. Die Prolog-Quelldatei wird in den so genannten WAM-Code kompiliert, welcher dann auf der WAM interpretiert wird. Die schnellsten Implementierungen einer WAM schaffen auf einem 1 Gigahertz-PC bis zu 10 Millionen logische Inferenzen pro Sekunde (LIPS).

5.2 Einfache Beispiele

Wir starten mit den Verwandtschaftsrelationen aus Beispiel 3.2. Die kleine Wissensbasis *WB* ist – ohne die Fakten zum Prädikat *weiblich* – in Abb. 5.1 als Prolog-Programm mit dem Namen *verw1.pl* kodiert.

Im Prolog-Interpreter kann das Programm mit dem Befehl

```
?- [verw1].
```

geladen und kompiliert werden. Eine erste Anfrage liefert den Dialog

```
?- kind(eva,otto,anna).
```

Yes

mit der korrekten Antwort Yes. Wie kommt diese Antwort zustande? Für die Anfrage „`?- kind(eva,otto,anna).`“ gibt es sechs Fakten und eine Regel mit dem gleichen Prädikat im Klauselkopf. Nun wird in der Reihenfolge des Vorkommens in der Eingabedatei versucht, die Anfrage mit jedem dieser komplementären Literale zu unifizieren. Scheitert eine der Alternativen, so führt dies zum **Backtracking** zurück zum letzten Verzweigungspunkt, und es wird die nächste Alternative getestet. Da die Unifikation mit allen Fakten scheitert, wird die Anfrage mit der rekursiven Regel in Zeile 8 unifiziert. Nun wird

Abb. 5.1 Prolog-Programm mit Verwandtschaftsbeziehungen

```

1 kind(otto,katrin,franz).
2 kind(maria,katrin,franz).
3 kind(eva,anna,otto).
4 kind(hans,anna,otto).
5 kind(isolde,anna,otto).
6 kind(klaus,maria,ottob).
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).

```

versucht, das Subgoal `kind(eva, anna, otto)` zu lösen, was bei der dritten Alternative erfolgreich ist. Die Anfrage

`?- nachkomme(X, Y).`

`X = otto`
`Y = katrin`

`Yes`

wird mit der ersten gefundenen Lösung beantwortet, genauso wie

`?- nachkomme(klaus, Y).`

`Y = maria`

`Yes`

Die Anfrage

`?- nachkomme(klaus, katrin).`

wird aber nicht beantwortet. Der Grund ist die Klausel in Zeile 8 für die Symmetrie des Kind-Prädikates. Diese Klausel ruft sich selbst rekursiv auf, aber ohne eine Möglichkeit der Terminierung. Gelöst wird dieses Problem durch folgendes neue Programm (aus Platzgründen wurden hier die Fakten 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).

```

Nun wird aber die Anfrage

`?- kind(eva, otto, anna).`

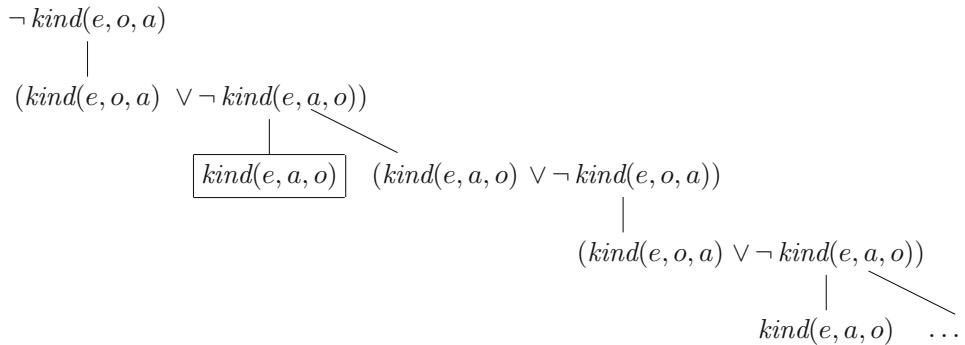


Abb. 5.2 Prolog-Suchbaum für `kind(eva, otto, anna)`

nicht mehr korrekt beantwortet, denn die Symmetrie von `kind` in den letzten beiden Variablen ist nicht mehr gegeben. Eine Lösung beider Probleme bringt das Programm

```

1  kind_fakt(otto,katrin,franz).
2  kind_fakt(maria,katrin,franz).
3  kind_fakt(eva,anna,otto).
4  kind_fakt(hans,anna,otto).
5  kind_fakt(isolde,anna,otto).
6  kind_fakt(klaus,maria,ottob).
7
8  kind(X,Z,Y) :- kind_fakt(X,Y,Z).
9  kind(X,Z,Y) :- kind_fakt(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 Prädikates `kind_fakt` für die Fakten ist das Prädikat `kind` nicht mehr rekursiv. Allerdings ist das Programm nicht mehr so elegant und einfach wie die – logisch korrekte – erste Variante in Abb. 5.1, die zu der Endlosschleife führt. Der Programmierer muß sich in Prolog, genauso wie bei jeder anderen Programmiersprache, um die Abarbeitung kümmern und Endlosschleifen vermeiden. Prolog ist eben eine Programmiersprache und kein Theorembeweiser.

Man muss hier unterscheiden zwischen der **deklarativen** und der **prozeduralen Semantik** von Prolog-Programmen. Die deklarative Semantik ist durch die logische Interpretation der Hornklauseln gegeben. Die prozedurale Semantik hingegen ist durch die Abarbeitung der Prolog-Programme definiert, die wir nun etwas genauer betrachten wollen. Die Abarbeitung des Programms aus Abb. 5.1 mit der Anfrage `kind(eva, otto, anna)` ist in Abb. 5.2 als Suchbaum dargestellt.¹ Die Abarbeitung startet links oben mit der Anfrage. Jede Kante repräsentiert einen möglichen SLD-Resolutionsschritt mit einem komplementen-

¹ Die Konstanten wurden aus Platzgründen abgekürzt.

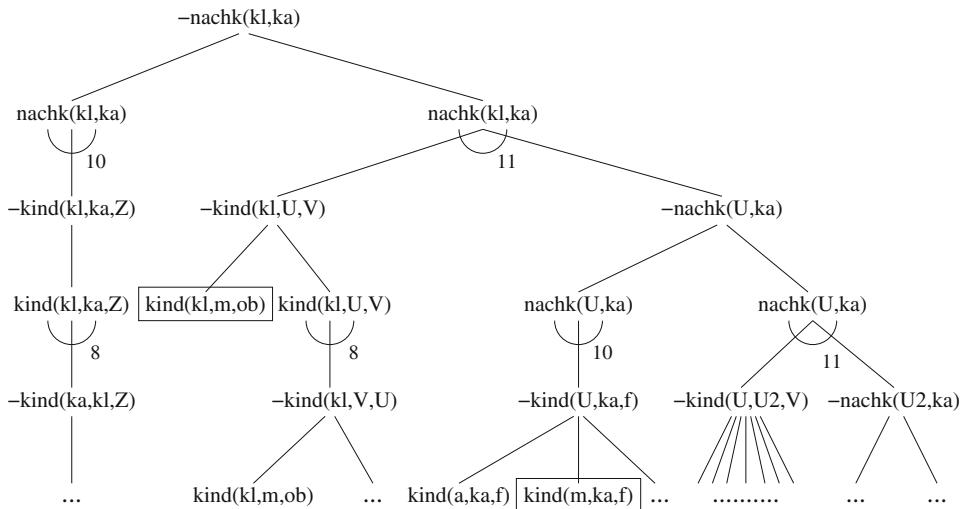


Abb. 5.3 Und-Oder-Baum für `nachk(klaus, katrin)`

tären unifizierbaren Literal. Obwohl der Suchbaum durch die rekursive Regel unendlich tief wird, terminiert die Prolog-Abarbeitung, denn die Fakten stehen in der Eingabedatei vor der Regel.

Bei der Anfrage `nachkomme(klaus, katrin)` hingegen terminiert die Prolog-Abarbeitung nicht. Dies erkennt man gut an dem in Abb. 5.3 dargestellten **Und-Oder-Baum**. Bei dieser Darstellung führen die durch dargestellten Verzweigungen vom Kopf einer Klausel zu den Teilzielen. Da zu einer Klausel immer alle Teilziele gelöst werden müssen, sind dies **Und-Verzweigungen**. Alle anderen Verzweigungen sind **Oder-Verzweigungen**, von denen mindestens eine mit dem Vorgängerknoten unifizierbar sein muß. Die beiden eingerahmten Fakten stellen die Lösung für die Anfrage dar. Der Prolog-Interpreter terminiert hier aber nicht, denn er arbeitet mittels Tiefensuche mit Backtracking (siehe Abschn. 6.2.2) und wählt daher zuerst den unendlich tiefen Pfad ganz links.

5.3 Ablaufsteuerung und prozedurale Elemente

Wie wir an dem Verwandtschaftsbeispiel gesehen haben, ist es wichtig, die Abarbeitung von Prolog zu kontrollieren. Insbesondere die Vermeidung von unnötigem Backtracking kann zu großen Effizienzsteigerungen führen. Ein Mittel hierzu ist der **Cut**. Durch das Einfügen eines Ausrufezeichens in eine Klausel kann das Backtracking über diese Stelle hinweg verhindert werden. In folgendem 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.

```

Trifft hier der erste Fall zu, so kann der zweite nicht mehr eintreffen. Trifft hingegen der erste Fall nicht zu, so ist automatisch die Bedingung der zweiten Klausel wahr, das heißt, sie muss nicht mehr überprüft werden. Zum Beispiel in der Anfrage

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

setzt wegen $Z = 3$ Backtracking ein und die zweite Klausel für `max` wird getestet, was zum Scheitern verurteilt ist. Daher ist das Backtracking über diese Stelle hinweg unnötig. Mit dem Cut kann man dies nun optimieren:

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

Damit wird die zweite Klausel nur dann aufgerufen, wenn es wirklich nötig ist; nämlich wenn die erste Klausel nicht zutrifft. Allerdings macht diese Optimierung das Programm schwerer verständlich.

Eine andere Möglichkeit der Ablaufsteuerung bietet das eingebaute Prädikat (engl. built-in Prädikat) `fail`, das nie wahr ist. Im Verwandtschaftsbeispiel kann man sich ganz einfach alle Kinder und deren Eltern ausgeben lassen mit der Anfrage

```
?- kind_fakt(X, Y, Z), write(X), write(' ist Kind von '), write(Y),
   write(' und '), write(Z), write('.'), nl, fail.
```

Die Ausgabe dazu lautet

```
otto ist Kind von katrin und franz.
maria ist Kind von katrin und franz.
eva ist Kind von anna und otto.
...
No.
```

wobei das Prädikat `nl` einen Zeilenumbruch in der Ausgabe bewirkt. Was würde ohne `fail` am Ende ausgegeben werden?

Bei der gleichen Wissensbasis würde die Anfrage „`?- kind_fakt(ulla, X, Y).`“ zur Antwort `No` führen, denn es gibt keine Fakten über `ulla`. Diese Antwort ist logisch nicht korrekt. Es ist nämlich nicht möglich zu beweisen, dass es kein Objekt mit dem Namen `ulla` gibt. Der Beweiser E würde hier korrekt antworten „`No proof found.`“ Wenn also Prolog mit `No` antwortet, so heißt dies nur, dass eine Anfrage Q nicht bewiesen werden kann. Dazu muss aber nicht notwendigerweise $\neg Q$ beweisbar sein. Dieses Verhalten nennt man **Negation as Failure**.

Die Einschränkung auf Hornklauseln stellt in den meisten Fällen kein großes Problem dar. Sie ist jedoch wichtig für die prozedurale Abarbeitung mittels SLD-Resolution (Abschn. 2.5). Durch das einzige festgelegte positive Literal pro Klausel hat die SLD-Resolution

und damit die Abarbeitung von Prolog-Programmen einen eindeutigen Einstiegspunkt in Klauseln. Nur so ist eine reproduzierbare Abarbeitung von Logikprogrammen und damit eine wohldefinierte prozedurale Semantik möglich.

Es gibt allerdings durchaus Aufgabenstellungen, die sich mit Hornklauseln nicht beschreiben lassen. Ein Beispiel ist die Russellsche Antinomie aus Beispiel 3.7, welche die Nicht-Hornklausel (*rasiert(barbier, X)* \vee *rasiert(X, X)*) enthält.

5.4 Listen

Prolog als Hochsprache hat wie auch die Sprache LISP den komfortabel benutzbaren generischen Datentyp der Liste. Eine Liste mit den Elementen *A, 2, 2, B, 3, 4, 5* hat die Gestalt

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

Das Konstrukt [Head | Tail] trennt das erste Element (Head) vom Rest (Tail) der Liste. Bei der Wissensbasis

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

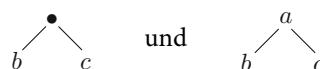
liefert Prolog den Dialog

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

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

Yes

Unter Verwendung von geschachtelten Listen kann man beliebige Baumstrukturen erzeugen. Zum Beispiel lassen sich die beiden Bäume



durch die Listen [b, c] beziehungsweise [a, b, c] darstellen oder die beiden Bäume



durch die Listen [[e, f, g], [h], d] beziehungsweise [a, [b, e, f, g], [c, h], d]. Bei den Bäumen mit Symbolen an den inneren Knoten enthält jeweils der Kopf der Liste das Symbol und der Rest sind die Nachfolgeknoten.

Ein schönes, elegantes Beispiel für die Listenverarbeitung ist die Definition des Prädikates `append(X, Y, Z)` zum Anhängen der Liste `Y` an die Liste `X`. Das Ergebnis wird in `Z` gespeichert. Das zugehörige Prolog-Programm lautet

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

Dies ist eine deklarative (rekursive) logische Beschreibung der Tatsache, dass `L3` durch Anhängen von `L2` an `L1` entsteht. Gleichzeitig macht 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, genauso wie der Aufruf

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

die Substitution `X = [4, 5, 6]` liefert. Hier erkennt man schön, dass `append` keine zweistellige Funktion, sondern eine dreistellige Relation darstellt. Man kann offenbar auch den „Ausgabeparameter“ `Z` eingeben und fragen, ob sich dieser erzeugen lässt.

Auch das Umkehren der Reihenfolge der Elemente einer Liste lässt sich sehr schön deklarativ beschreiben und gleichzeitig programmieren durch das rekursive Prädikat

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

welches das Umkehren einer Liste zurückführt auf das Umkehren der um ein Element kürzeren Liste. Allerdings ist dieses Prädikat wegen des Aufrufs von `append` sehr ineffizient. Dieses Programm ist unter dem Namen **Naive Reverse** bekannt und wird oft als Prolog-Benchmark verwendet (siehe Aufgabe 5.6). Besser geht es, wenn man unter Verwendung eines Zwischenspeichers, des so genannten Akkumulators, folgendermaßen vorgeht:

Liste	Akkumulator
[a,b,c,d]	[]
[b,c,d]	[a]
[c,d]	[b,a]
[d]	[c,b,a]
[]	[d,c,b,a]

Das zugehörige Prolog-Programm lautet

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

5.5 Selbstmodifizierende Programme

Prolog-Programme werden nicht voll kompiliert, sondern von der WAM interpretiert. Daraus ist es möglich, zur Laufzeit Programme zu verändern. Ein Programm kann sich sogar selbst verändern. Mit Befehlen wie zum Beispiel `assert` und `retract` können Fakten und Regeln zur Wissensbasis hinzugefügt oder aus ihr entfernt werden.

Eine einfache Anwendung der Variante `asserta` ist das Hinzufügen von abgeleiteten Fakten an den Anfang der Wissensbasis mit dem Ziel, ein wiederholtes, eventuell zeitaufwändiges Ableiten zu vermeiden (siehe Aufgabe 5.8). Ersetzt man in unserem Verwandtschaftsbeispiel die beiden Regeln für das Prädikat Nachkomme durch

```
1  :- dynamic nachkomme/2.  
2  nachkomme(X,Y) :- kind(X,Y,Z), asserta(nachkomme(X,Y)).  
3  nachkomme(X,Y) :- kind(X,U,V), nachkomme(U,Y),  
4                                asserta(nachkomme(X,Y)).
```

so werden alle abgeleiteten Fakten zu diesem Prädikat in der Wissensbasis gespeichert und somit in Zukunft nicht nochmal abgeleitet. Die Anfrage

```
?- nachkomme(klaus, katrin).
```

führt zum Hinzufügen der beiden Fakten

```
nachkomme(klaus, katrin).  
nachkomme(maria, katrin).
```

Durch die Manipulation von Regeln mit `assert` und `retract` können sogar Programme geschrieben werden, die sich selbst komplett verändern. Unter dem Begriff **Genetic Programming** wurde diese Idee bekannt und ermöglicht den Bau von beliebig flexibel lernfähigen Programmen. In der Praxis hat sich jedoch gezeigt, dass Programme, die durch simples versuchsweises Verändern ihres eigenen Codes aufgrund der immens großen Zahl von unsinnigen Veränderungsmöglichkeiten praktisch nie zu einer Verbesserung der Leistungsfähigkeit kommen. Das systematische Verändern von Regeln hingegen macht die Programmierung dermaßen komplex, dass bis heute derartige Programme, die ihren eigenen Code virtuos verändern, nicht erfolgreich waren. Im Kap. 8 werden wir aufzeigen, inwiefern das Maschinelle Lernen durchaus zu Erfolgen geführt hat. Allerdings werden hier nur sehr eingeschränkte automatische Veränderungen am Programmcode vorgenommen.

5.6 Ein Planungsbeispiel

Beispiel 5.1

Folgendes Rätsel dient uns als Aufgabenstellung für ein typisches Prolog-Programm.

Ein Bauer wollte einen Kohlkopf, eine Ziege und einen Wolf über einen Fluss bringen. Sein Boot war aber so klein, dass er entweder nur den Kohlkopf oder nur die Ziege oder nur den Wolf hinüberfahren konnte. Der Bauer dachte nach und sagte dann zu sich: „Bringe ich zuerst den Wolf ans andere Ufer, so frisst die Ziege den Kohl. Transportiere ich den Kohl als erstes, wird die Ziege vom Wolf gefressen. Was soll ich tun?“

Dies ist eine Planungsaufgabe, die man mit etwas Nachdenken schnell löst. Nicht ganz so schnell ist das in Abb. 5.4 angegebene Prolog-Programm erstellt.

Das Programm arbeitet auf Termen der Form `zust(Bauer,Wolf,Ziege,Kohl)`, welche den aktuellen Zustand der Welt beschreiben. Die vier Variablen mit den möglichen Werten `links`, `rechts` geben den Ort der Objekte an. Das zentrale rekursive Prädikat `plan` erzeugt zuerst mit `gehe` einen Nachfolgezustand `Next`, testet mit `sicher` dessen Sicherheit und wiederholt dies rekursiv bis Start- und Zielzustand gleich sind (in Programmzeile 15). Im dritten Argument von `plan` wird die Liste der schon besuchten Zustände gespeichert. Mit dem Built-in-Prädikat `member` wird geprüft, ob der Zustand `Next` schon besucht wurde. Wenn ja, wird dieser verworfen.

Die Definition des Prädikates `write_path` zur Ausgabe des gefundenen Plans fehlt hier. Sie wird dem Leser zur Übung empfohlen (Aufgabe 5.2). Für erste Tests des Programms kann das Literal `write_path(Pfad)` ersetzt werden durch `write(Pfad)`. Auf die Anfrage „?- `start.`“ erhalten wir die Antwort

Loesung:

```
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
```

Yes

Zum besseren Verständnis wollen wir die Definition von `plan` in Logik beschreiben:

$$\forall z \text{plan}(z,z) \wedge \forall s \forall z \forall n [\text{gehe}(s,n) \wedge \text{sicher}(n) \wedge \text{plan}(n,z) \Rightarrow \text{plan}(s,z)]$$

Diese Definition fällt wesentlich knapper aus als in Prolog. Hierfür gibt es zwei Gründe. Zum einen ist für die Logik eines Plans die Ausgabe des gefundenen Pfades unwich-

```

1  start :- aktion(zust(links,links,links,links),
2                  zust(rechts,rechts,rechts,rechts)).
3
4  aktion(Start,Ziel):-
5      plan(Start,Ziel,[Start],Pfad),
6      nl,write('Loesung:'),nl,
7      write_path(Pfad).
8
9  plan(Start,Ziel,Besucht,Pfad):-
10     gehe(Start,Next),
11     sicher(Next),
12     \+ member(Next,Besucht),          % not(member(...))
13     plan(Next,Ziel,[Next|Besucht],Pfad).
14 plan(Ziel,Ziel,Pfad,Pfad).
15
16 gehe(zust(X,X,Z,K),zust(Y,Y,Z,K)):-gegenueber(X,Y). % Bauer, Wolf
17 gehe(zust(X,W,X,K),zust(Y,W,Y,K)):-gegenueber(X,Y). % Bauer, Ziege
18 gehe(zust(X,W,Z,X),zust(Y,W,Z,Y)):-gegenueber(X,Y). % Bauer, Kohl
19 gehe(zust(X,W,Z,K),zust(Y,W,Z,K)):-gegenueber(X,Y). % Bauer
20
21 gegenueber(links,rechts).
22 gegenueber(rechts,links).
23
24 sicher(zust(B,W,Z,K)):- gegenueber(W,Z), gegenueber(Z,K).
25 sicher(zust(B,B,B,K)).
26 sicher(zust(B,W,B,B)).
```

Abb. 5.4 Prolog-Programm für das Bauer-Wolf-Ziege-Kohl-Problem

tig. Außerdem ist die Überprüfung, ob der Nachfolgezustand schon besucht wurde, nicht wirklich nötig, wenn den Bauer unnötige Fahrten nicht stören. Wird jedoch im Prolog-Programm `member (...) weggelassen`, so entsteht eine Endlosschleife und Prolog findet eventuell keinen Plan, auch wenn es einen gibt. Ursache hierfür ist die Backward-Chainig-Suchstrategie von Prolog, die nach dem Prinzip der Tiefensuche (Abschn. 6.2.2) immer ein Subgoal nach dem anderen abarbeitet ohne Beschränkung der Rekursionstiefe und damit unvollständig ist. Einem Theorembeweiser mit vollständigem Kalkül würde dies nicht passieren.

Wie bei allen Planungsaufgaben verändert sich der Zustand der Welt durch das Ausführen von Aktionen von einem Schritt zum nächsten. Daher ist es naheliegend, in allen Prädikaten, die vom aktuellen Zustand der Welt abhängen, den Zustand als Variable mitzuführen, wie zum Beispiel in dem Prädikat `sicher`. Die Zustandsübergänge erfolgen in dem Prädikat `gehe`. Diese Vorgehensweise wird als **Situationskalkül** bezeichnet [RN03]. Eine interessante Erweiterung auf das Lernen von Aktionssequenzen in teilweise beobachtbaren nichtdeterministischen Welten werden wir in Kap. 10 kennenlernen.

5.7 Constraint Logic Programming

Die Programmierung von Terminplanungssystemen, bei denen viele, teilweise komplexe logische oder auch numerische Bedingungen erfüllt sein müssen, kann mit konventionellen Programmiersprachen sehr aufwändig und schwierig werden. Hier bietet sich die Logik geradezu an. Man schreibt einfach alle logischen Bedingungen in PL1 auf und stellt dann eine Anfrage. Meist scheitert man mit diesem Ansatz jedoch kläglich. Der Grund ist das in Abschn. 4.3 schon diskutierte Pinguin-Problem. Das Fakt `pinguin(tweety)` stellt zwar sicher, dass `pinguin(tweety)` wahr ist. Es schließt aber nicht aus, dass `rabe(tweety)` auch gilt. Dies kann nur sehr umständlich mit zusätzlichen Axiomen (Abschn. 4.3) ausgeschlossen werden.

Einen eleganten und sehr effizienten Mechanismus zur Lösung dieses Problems bietet das **Constraint Logic Programming (CLP)**, welches die explizite Formulierung von Randbedingungen (engl. constraint) für Variablen erlaubt. Der Interpreter überwacht während der Ausführung des entsprechenden Programms jederzeit die Einhaltung aller Randbedingungen. Der Programmierer wird von der Aufgabe, die Randbedingungen zu kontrollieren, vollkommen entlastet, was in vielen Fällen die Programmierung stark vereinfacht. Dies kommt auch in folgendem Zitat von Eugene C. Freuder aus [Fre97] zum Ausdruck:

Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.

Ohne auf die Theorie des Constraint Satisfaction Problems (CSP) einzugehen, wenden wir nun auf folgendes Beispiel den CLP-Mechanismus von GNU-Prolog [Dia04] an.

Beispiel 5.2

Für die Raumaufteilung am Albert-Einstein-Gymnasium bei den mündlichen Abiturprüfungen soll der Hausmeister einen Plan erstellen. Er hat folgende Informationen: Die vier Lehrer Maier, Huber, Müller und Schmid prüfen die Fächer Deutsch, Englisch, Mathe und Physik in den aufsteigend nummerierten Räumen 1, 2, 3 und 4. Jeder Lehrer prüft genau ein Fach in genau einem Raum. Außerdem weiß er folgendes über die Lehrer und ihre Fächer:

1. Herr Maier prüft nie in Raum 4.
2. Herr Müller prüft immer Deutsch.
3. Herr Schmid und Herr Müller prüfen nicht in benachbarten Räumen.
4. Frau Huber prüft Mathematik.
5. Physik wird immer in Raum 4 geprüft.
6. Deutsch und Englisch werden nicht in Raum 1 geprüft.

Wer prüft was in welchem Raum?

```

1   start :- 
2       fd_domain([Maier, Huber, Mueller, Schmid],1,4),
3       fd_all_different([Maier, Mueller, Huber, Schmid]),
4
5       fd_domain([Deutsch, Englisch, Mathe, Physik],1,4),
6       fd_all_different([Deutsch, Englisch, Mathe, Physik]),
7
8       fd_labeling([Maier, Huber, Mueller, Schmid]),
9
10      Maier #\= 4,                      % Maier prüft nicht in Raum 4
11      Mueller #= Deutsch,             % Müller prüft Deutsch
12      dist(Mueller,Schmid) #>= 2,    % Abstand Müller/Schmid >= 2
13      Huber #= Mathe,                % Huber prüft Mathematik
14      Physik #= 4,                   % Physik in Raum 4
15      Deutsch #\= 1,                 % Deutsch nicht in Raum 1
16      Englisch #\= 1,                % Englisch nicht in Raum 1
17      nl,
18      write([Maier, Huber, Mueller, Schmid]), nl,
19      write([Deutsch, Englisch, Mathe, Physik]), nl.
```

Abb. 5.5 CLP-Programm für das Raumplanproblem

In Abb. 5.5 ist ein GNU-Prolog-Programm zur Lösung dieses Problems angegeben. Dieses Programm arbeitet mit den Variablen Maier, Huber, Mueller, Schmid sowie Deutsch, Englisch, Mathe, Physik, welche jeweils die ganzzahligen Werte 1 bis 4 als Raumnummern annehmen können (Programmzeilen 2 und 5). Eine Belegung Maier = 1 und Deutsch = 1 bedeutet, dass Herr Maier in Raum 1 Deutsch prüft. Die Zeilen 3 und 6 stellen sicher, dass die jeweiligen vier Variablen verschiedene Werte annehmen. Zeile 8 stellt sicher, dass im Fall einer Lösung allen Variablen ein konkreter Wert zugeordnet wird. Diese Zeile ist hier nicht unbedingt erforderlich. Wenn es jedoch mehrere Lösungen gibt, werden ohne diese Zeile eventuell nur Intervalle ausgegeben. In den Zeilen 10 bis 16 sind die Randbedingungen angegeben, und die restlichen Zeilen geben in einfacher Form die Raumnummern für alle Lehrer und alle Fächer aus.

Mit „['raumplan.pl'].“ wird das Programm in GNU-Prolog geladen und mit „start.“ erhält man die Ausgabe

```
[3,1,2,4]
[2,3,1,4]
```

Etwas komfortabler dargestellt ergibt sich folgender Raumplan

Raum Nr.	1	2	3	4
Lehrer	Huber	Müller	Maier	Schmid
Fach	Mathe	Deutsch	Englisch	Physik

GNU-Prolog besitzt, wie die meisten anderen CLP-Sprachen auch, einen so genannten **Finite-Domain-Constraint-Solver**, bei dem den Variablen ein endlicher Wertebereich aus den ganzen Zahlen zugeordnet werden kann. Dies muss nicht unbedingt ein Intervall sein wie im Beispiel. Man kann auch eine Liste von Werten angeben. Zur Übung wird der Leser aufgefordert, in Aufgabe 5.9 zum Beispiel mit GNU-Prolog ein CLP Programm für ein gar nicht so einfaches Logikrätsel zu erstellen. Dieses angeblich von Einstein stammende Rätsel kann mit einem CLP-System ganz einfach gelöst werden. Mit Prolog ohne Constraints hingegen kann man sich daran die Zähne ausbeißen. Wer mit Prolog oder einem Beweiser eine elegante Lösung findet, der möge diese bitte dem Autor zukommen lassen.

5.8 Zusammenfassung

Unifikation, Listen, deklarative Programmierung, sowie die relationale Sicht auf Prozeduren, bei denen ein Übergabeparameter sowohl Eingabe- als auch Ausgabeparameter sein kann, erlauben für viele Problemstellungen den Entwurf eleganter kurzer Programme. In prozeduralen Sprachen wären viele Programme wesentlich länger und damit schwieriger verständlich. Außerdem spart der Programmierer Zeit. Deshalb ist Prolog auch ein interessantes Werkzeug zum Rapid Prototyping speziell für KI-Anwendungen. Sehr hilfreich nicht nur für Logikrätsel, sondern auch für viele Optimierungs- und Planungsaufgaben ist die vorgestellte CLP-Erweiterung von Prolog.

Seit der Erfindung im Jahr 1972 hat sich Prolog in Europa neben den prozeduralen Sprachen zur führenden Programmiersprache in der KI entwickelt. In den USA hingegen dominiert die dort erfundene Sprache LISP immer noch den Markt in der KI.

Prolog ist kein Theorembeweiser. Dies ist so gewollt, denn ein Programmierer muss die Abarbeitung einfach und flexibel steuern können. Er wird mit einem Theorembeweiser nicht weit kommen. Umgekehrt wird Prolog alleine nicht sehr hilfreich sein, wenn mathematische Sätze bewiesen werden sollen. Es gibt aber durchaus interessante Theorembeweiser, die in Prolog programmiert sind.

Als weiterführende Literatur empfehlen sich [Bra86] und [CM94] sowie die Handbücher [Wie04, Dia04] und zum Thema CLP [Bar98].

5.9 Übungen

Aufgabe 5.1 Versuchen Sie, den Satz aus Abschn. 3.7 über die Gleichheit von links- und rechtsneutralen Element von Halbgruppen mit Prolog zu beweisen. Welche Probleme treten auf? Wo liegt die Ursache?

Aufgabe 5.2

- a) Schreiben Sie ein Prädikat `write_move (+Zust1, +Zust2)`, das für jede Bootsfahrt des Bauern aus Beispiel 5.1 einen Satz wie etwa „Bauer und Wolf fahren von links nach rechts“ ausgibt. Zust1 und Zust2 sind Terme der Form `zust (Bauer, Wolf, Ziege, Kohl)`.
- b) Schreiben Sie ein rekursives Prädikat `write_path (+Pfad)`, welches das Prädikat `write_move (+Zust1, +Zust2)` aufruft und alle Aktionen des Bauern ausgibt.

Aufgabe 5.3

- a) Auf den ersten Blick ist die Variable Pfad im Prädikat `plan` des Prolog-Programms von Beispiel 5.1 unnötig, denn sie wird scheinbar nirgends verändert. Wozu wird sie benötigt?
- b) Ergänzt man im Beispiel die Definition von `aktion` um ein `fail` am Ende, so werden alle Lösungen ausgegeben. Warum wird nun aber jede Lösung zweimal ausgegeben? Wie können Sie dies verhindern?

Aufgabe 5.4

- a) Zeigen Sie durch Ausprobieren, dass der Theorembeweiser E (im Gegensatz zu Prolog) mit der Wissensbasis aus Abb. 5.1 die Anfrage
`? - nachkomme (klaus, katrin).`
richtig beantwortet. Woran liegt das?
- b) Vergleichen Sie die Antworten von Prolog und E auf die Anfrage
`? - nachkomme (X, Y).`

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

Aufgabe 5.6 \Rightarrow Das Laufzeitverhalten des Naive-Reverse Prädikates soll untersucht werden.

- a) Lassen Sie Prolog mit der Trace-Option laufen und beobachten Sie die rekursiven Aufrufe von `nrev`, `append` und `accrev`.
- b) Berechnen Sie die asymptotische Zeitkomplexität von `append (L1, L2, L3)`, d.h. die Abhängigkeit der Rechenzeit von der Länge der Listen bei großen Listen. Nehmen Sie an, der Zugriff auf den Kopf einer beliebig langen Liste erfolgt in konstanter Zeit.
- c) Berechnen Sie die Zeitkomplexität von `nrev (L, R)`.
- d) Berechnen Sie die Zeitkomplexität von `accrev (L, R)`.
- e) Bestimmen Sie experimentell die Zeitkomplexität der Prädikate `nrev`, `append` und `accrev`, indem Sie z. B. mit SWI-Prolog Zeitmessungen durchführen (`time (+Goal)` gibt Inferenzen und CPU-Zeit an).

Aufgabe 5.7 Verwenden Sie Funktionssymbole statt Listen, um die in Abschn. 5.4 in Abschn. 5.4 angegebenen Bäume darzustellen.

Aufgabe 5.8 \Rightarrow Die Fibonaccifolge ist rekursiv definiert durch $fib(0) = 1$, $fib(1) = 1$ und $fib(n) = fib(n - 1) + fib(n - 2)$.

- Definieren Sie ein rekursives Prolog-Prädikat `fib(N, R)`, das $fib(N)$ berechnet und in `R` zurückgibt.
- Bestimmen Sie theoretisch und durch Messungen die Laufzeitkomplexität des Prädikates `fib`.
- Ändern Sie unter Verwendung von `asserta` Ihr Programm so ab, dass keine unnötigen Inferenzen mehr ausgeführt werden.
- Bestimmen Sie theoretisch und durch Messungen die Laufzeitkomplexität des geänderten Prädikates (beachten Sie, dass diese davon abhängt, ob `fib` schon vorher aufgerufen wurde).
- Warum ist `fib` mit `asserta` auch dann schneller, wenn es zum ersten mal nach dem Start von Prolog gestartet wird?

Aufgabe 5.9 \Rightarrow Das folgende typische Logikrätsel soll angeblich Albert Einstein verfasst haben. Außerdem soll er behauptet haben, dass nur 2 % der Weltbevölkerung in der Lage seien, es zu lösen. Gegeben sind die Aussagen:

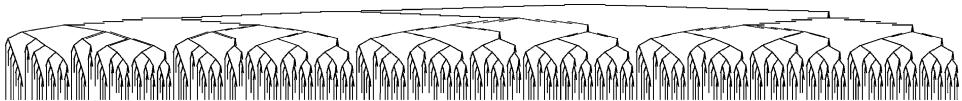
- Es gibt 5 Häuser mit je einer anderen Farbe.
- In jedem Haus wohnt eine Person einer anderen Nationalität.
- Jeder Hausbewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke und hält ein bestimmtes Haustier.
- Keine der 5 Personen trinkt das Gleiche, raucht das Gleiche oder hält das gleiche Tier.
- Hinweise:
 - Der Brite lebt im roten Haus.
 - Der Schwede hält einen Hund.
 - Der Däne trinkt gerne Tee.
 - Das grüne Haus steht links vom weißen Haus.
 - Der Besitzer vom grünen Haus trinkt Kaffee.
 - Die Person, die Pall Mall raucht, hält 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 Marlbororaucher wohnt neben dem, der eine Katze hält.
 - Der Mann, der ein Pferd hält, wohnt neben dem, der Dunhill raucht.
 - Der Winfieldraucher trinkt gerne Bier.
 - Der Norweger wohnt neben dem blauen Haus.
 - Der Deutsche raucht Rothmanns.
 - Der Marlbororaucher hat einen Nachbarn, der Wasser trinkt.

Frage: Wem gehört der Fisch?

- a) Lösen Sie das Rätsel zuerst manuell.
- b) Schreiben Sie ein CLP-Programm (z. B. mit GNU-Prolog) zur Lösung des Rätsels. Orientieren Sie sich an dem Raumplanungsprogramm in Abb. 5.5.

6.1 Einführung

Bei fast allen Inferenzsystemen stellt die Suche nach einer Lösung, bedingt durch die extrem großen Suchbäume, ein Problem dar. Aus dem Startzustand gibt es für den ersten Inferenzschritt viele Möglichkeiten. Für jede dieser Möglichkeiten gibt es im nächsten Schritt wieder viele Möglichkeiten und so weiter. Schon beim Beweis einer ganz einfachen Formel aus [Ert93] mit drei Hornklauseln mit maximal drei Literalen hat der Suchbaum für SLD-Resolution folgende Gestalt:



Der Baum wurde bei einer Tiefe von 14 abgeschnitten und besitzt in dem mit * markierten Blattknoten eine Lösung. Nur durch den kleinen Verzweigungsfaktor von maximal zwei und das Abschneiden des Suchbaumes auf Tiefe 14 ist er überhaupt darstellbar. Bei realistischen Problemen können Verzweigungsfaktor und Tiefe der ersten Lösung deutlich größer werden.

Angenommen, der Verzweigungsfaktor ist konstant gleich 30 und die erste Lösung liegt in Tiefe 50. Dann besitzt der Suchbaum $30^{50} \approx 7,2 \cdot 10^{73}$ Blattknoten. Die Zahl der Inferenzschritte ist aber noch größer, denn nicht nur jeder Blattknoten, sondern auch jeder innere Knoten des Baumes entspricht einem Inferenzschritt. Wir müssen also über alle Ebenen des Baumes die Zahl der Knoten addieren und erhalten als Gesamtzahl der Knoten des Suchbaumes

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

was nicht viel an der Zahl der Knoten ändert. Offenbar liegen fast alle Knoten dieses Suchbaumes auf der letzten Ebene. Wie wir sehen werden, gilt dies ganz allgemein. Nun aber zurück zu dem Suchbaum mit den $7,4 \cdot 10^{73}$ Knoten. Angenommen wir hätten 10.000 Computer, von denen jeder eine Milliarde Inferenzen pro Sekunde schaffen würde und wir könnten die Arbeit ohne Verluste auf alle Rechner verteilen. Die gesamte Rechenzeit für alle $7,4 \cdot 10^{73}$ Inferenzen wäre dann etwa gleich

$$\frac{7,4 \cdot 10^{73} \text{ Inferenzen}}{10.000 \cdot 10^9 \text{ Inferenzen/s}} = 7,4 \cdot 10^{60} \text{ s} \approx 2,3 \cdot 10^{53} \text{ Jahre},$$

was wiederum etwa 10^{43} mal so lange dauert wie unser Universum alt ist. Man erkennt also durch diese einfache Überlegung schnell, dass mit den uns auf dieser Welt zur Verfügung stehenden Mitteln keine realistische Chance besteht, derartige Suchräume komplett zu durchsuchen. Die Annahme bezüglich der Größe des Suchraumes war übrigens durchaus realistisch. Beim Schachspiel zum Beispiel gibt es für eine typische Stellung über 30 mögliche Züge, und eine Spieldauer von 50 Halbzügen ist noch relativ kurz.

Wie kann es dann sein, dass es gute Schachspieler – und heutzutage auch gute Schachcomputer – gibt? Wie kann es sein, dass Mathematiker Beweise für Sätze finden, bei denen der Suchraum noch viel größer ist? Offenbar verwenden wir Menschen intelligente Strategien, die den Suchraum drastisch reduzieren. Der erfahrene Schachspieler, genauso wie der erfahrene Mathematiker, wird durch bloßes Betrachten der Aufgabenstellung viele der möglichen Aktionen sofort als unsinnig ausschließen. Durch seine Erfahrung besitzt er die Fähigkeit, die verschiedenen Aktionen bezüglich ihres Nutzens für das Erreichen des Ziels zu bewerten. Oft geht der Mensch hier gefühlsmäßig vor. Wenn man einen Mathematiker fragt, wie er einen Beweis gefunden hat, wird er eventuell antworten, dass ihm die Eingabe im Traum erschien. Viele Ärzte finden in schwierigen Fällen die Diagnose rein gefühlsmäßig, basierend auf allen bekannten Symptomen. Gerade bei schwierigen Aufgabenstellungen gibt es oft keine formale Theorie der Lösungsfindung, welche eine optimale Lösung garantiert. Auch bei ganz alltäglichen Problemen wie zum Beispiel der Suche nach der entlaufenen Katze in Abb. 6.1 spielt die Intuition eine große Rolle. Derartige **heuristische Suchverfahren** werden wir in Abschn. 6.3 behandeln und außerdem Verfahren beschreiben, mit denen Computer, ähnlich wie wir Menschen, ihre heuristischen Suchstrategien durch Lernen verbessern können.

Zuvor jedoch müssen wir verstehen, wie die **uninformierte Suche**, das heißt das blinde Durchprobieren aller Möglichkeiten, funktioniert. Wir starten mit einigen Beispielen.

Beispiel 6.1

Am 8-Puzzle, dem klassischen Beispiel für Suchalgorithmen [Nil98, RN03] lassen sich die verschiedenen Algorithmen sehr anschaulich erläutern. Auf einer 3×3 Matrix wie in Abb. 6.2 sind 8 Plättchen mit den Nummern 1 bis 8 verteilt. Ziel ist es, eine bestimmte Anordnung der Plättchen, zum Beispiel die in Abb. 6.2 rechts dargestellte, aufsteigend sortierte zeilenweise Anordnung, zu erreichen. In jedem Schritt kann ein Plättchen um

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

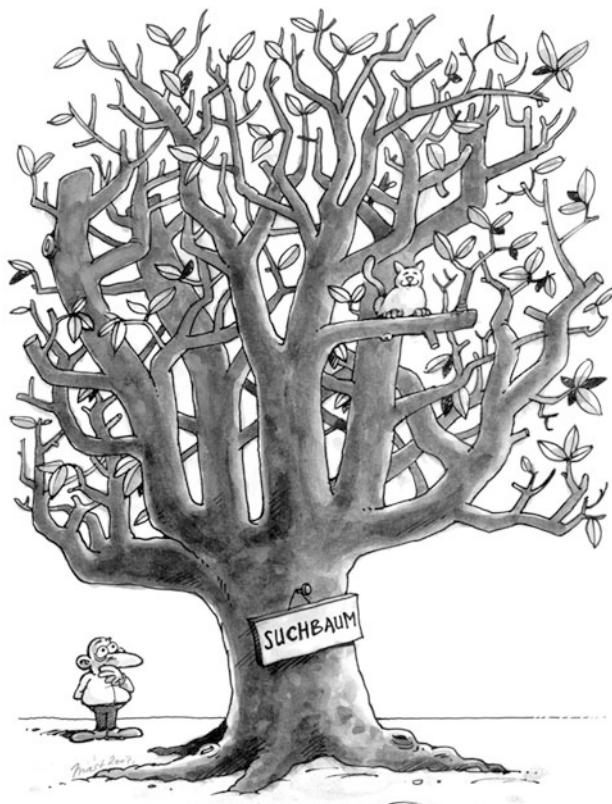
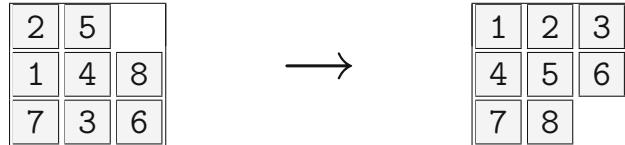


Abb. 6.2 Mögliche Start- und Zielzustände des 8-Puzzle



ein Feld nach links, rechts, oben oder unten auf das leere Feld verschoben werden. Das leere Feld bewegt sich dann in jeweils umgekehrter Richtung. Es bietet sich an, zur Analyse des Suchbaumes immer die möglichen Bewegungen des leeren Feldes zu betrachten.

In Abb. 6.3 ist der Suchbaum für einen Startzustand dargestellt. Man erkennt gut, dass der Verzweigungsfaktor zwischen zwei, drei und vier wechselt. Gemittelt über jeweils zwei Ebenen erhält man einen mittleren Verzweigungsfaktor¹ von $\sqrt{8} \approx 2,83$. Man erkennt, dass jeder Zustand zwei Ebenen tiefer mehrfach wiederholt auftritt, denn

¹ Der mittlere Verzweigungsfaktor eines Baumes ist der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleich vielen Blattknoten hätte.

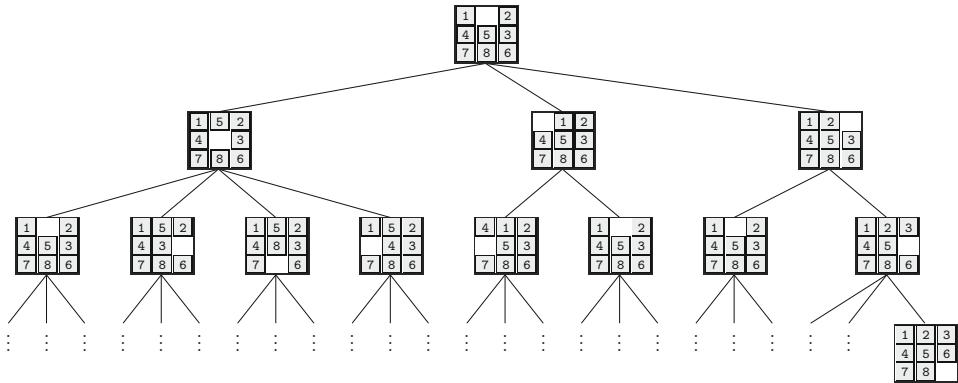


Abb. 6.3 Suchbaum beim 8-Puzzle. Rechts unten ist ein Zielzustand in Tiefe 3 dargestellt. Aus Platzgründen wurden die anderen Knoten auf dieser Ebene weggelassen

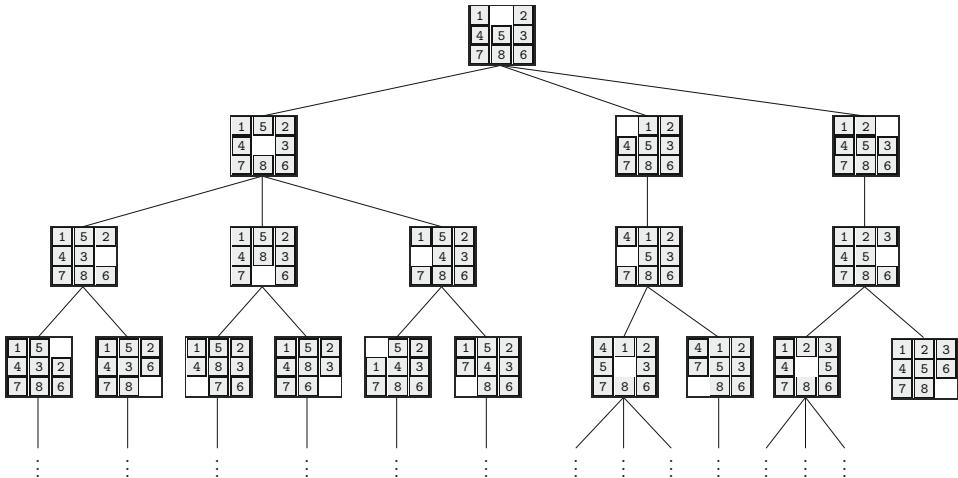


Abb. 6.4 Suchbaum beim 8-Puzzle ohne Zyklen der Länge 2

bei der einfachen uninformierten Suche kann jede Aktion im nächsten Schritt wieder rückgängig gemacht werden.

Verhindert man Zyklen der Länge 2, so erhält man für den gleichen Startzustand den in Abb. 6.4 dargestellten Suchbaum. Der mittlere Verzweigungsfaktor reduziert sich dadurch etwa um 1 auf 1,8.²

Bevor wir mit der Beschreibung der **Suchalgorithmen** beginnen, werden noch einige Begriffe benötigt. Wir behandeln hier diskrete Suchprobleme. Ausgehend von einem Zu-

² Beim 8-Puzzle hängt der mittlere Verzweigungsfaktor vom Startzustand ab (siehe Aufgabe 6.2).

stand s führt eine **Aktion** a_1 in einen neuen Zustand s' . Es gilt also $s' = a_1(s)$. Eine andere Aktion kann in einen anderen Zustand s'' führen, das heißt $s'' = a_2(s)$. Durch die rekursive Anwendung aller möglichen Aktionen auf alle Zustände, beginnend mit dem Startzustand, entsteht der **Suchbaum**.

Definition 6.1

Ein Suchproblem wird definiert durch folgende Größen

- Zustand:** Beschreibung des Zustands der Welt, in dem sich ein Suchagent befindet.
- Startzustand:** Der Initialzustand, in dem der Agent gestartet wird.
- Zielzustand:** Erreicht der Agent einen Zielzustand, so terminiert er und gibt (falls gewünscht) eine Lösung aus.
- Aktionen:** Alle erlaubten Aktionen des Agenten.
- Lösung:** Der Pfad im Suchbaum vom Startzustand zum Zielzustand.
- Kostenfunktion:** Ordnet jeder Aktion einen Kostenwert zu. Wird benötigt, um kostenoptimale Lösungen zu finden.
- Zustandsraum:** Menge aller Zustände.

Angewandt auf das 8-Puzzle-Problem ergibt sich

- Zustand:** 3×3 Matrix S mit den Werten 1, 2, 3, 4, 5, 6, 7, 8 (je einmal) und einem leeren Feld.
- Startzustand:** Ein beliebiger Zustand.
- Zielzustand:** Ein beliebiger Zustand, z. B. der in Abb. 6.2 rechts angegebene Zustand.
- Aktionen:** Bewegung des leeren Feldes S_{ij} nach links (falls $j \neq 1$), rechts (falls $j \neq 3$), oben (falls $i \neq 1$), unten (falls $i \neq 3$).
- Kostenfunktion:** Die konstante Funktion 1, da alle Aktionen gleich aufwändig sind.
- Zustandsraum:** Der Zustandsraum zerfällt in Bereiche, die gegenseitig nicht erreichbar sind (Aufgabe 6.4). Daher gibt es nicht lösbare 8-Puzzle-Probleme.

Zur Analyse der Suchalgorithmen werden noch folgende Begriffe benötigt:

Definition 6.2

- Die Zahl der Nachfolgezustände eines Zustands s wird als **Verzweigungsfaktor** (engl. branching faktor) $b(s)$ bezeichnet, beziehungsweise mit b , falls der Verzweigungsfaktor konstant ist.

- Der **effektive Verzweigungsfaktor** eines Baumes der Tiefe d mit insgesamt n Knoten wird definiert als der Verzweigungsfaktor, den ein Baum mit konstantem Verzweigungsfaktor, gleicher Tiefe und gleichem n hätte (siehe Aufgabe 6.3).
- Ein Suchalgorithmus heißt **vollständig**, wenn er für jedes lösbare Problem eine Lösung findet. Terminiert ein vollständiger Suchalgorithmus ohne eine Lösung zu finden, so ist das Problem also nicht lösbar.

Bei gegebener Tiefe d und Knotenzahl n lässt sich der effektive Verzweigungsfaktor durch Auflösen der Gleichung

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

nach b berechnen, denn ein Baum mit konstantem Verzweigungsfaktor und Tiefe d hat insgesamt

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

Knoten.

Für die praktische Anwendung von Suchalgorithmen ist bei endlichen Suchbäumen die letzte Ebene besonders wichtig, denn es gilt

Satz 6.1

Bei stark verzweigenden endlichen Suchbäumen mit großem konstantem Verzweigungsfaktor liegen fast alle Knoten auf der letzten Ebene.

Der einfache Beweis dieses Satzes wird dem Leser als Übung empfohlen (Aufgabe 6.1).

Beispiel 6.2

Gegeben ist eine Landkarte wie in Abb. 6.5 dargestellt, als Graph mit Städten als Knoten und Autobahnverbindungen zwischen den Städten als gewichtete Kanten mit Entfernung. Gesucht ist eine möglichst optimale Route von einer Stadt A zu Stadt B . Die Beschreibung entsprechend dem Schema lautet nun:

Zustand: Eine Stadt als aktueller Ort des Reisenden.

Startzustand: Eine beliebige Stadt.

Zielzustand: Eine beliebige Stadt.

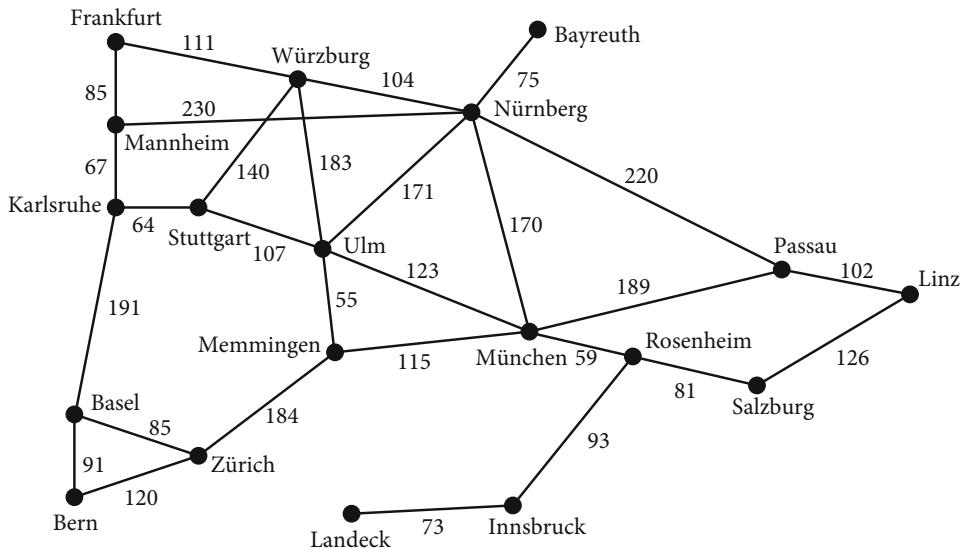


Abb. 6.5 Der Süddeutschlandgraph als Beispiel für eine Suchaufgabe mit Kostenfunktion

Aktionen: Reise von der aktuellen Stadt zu einer Nachbarstadt.

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

Zustandsraum: Alle Städte, d. h. Knoten im Graphen.

Um eine Route mit minimaler Länge zu finden, müssen hier die Kosten berücksichtigt werden, denn sie sind nicht wie beim 8-Puzzle konstant.

Definition 6.3

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

Das 8-Puzzle-Problem ist **deterministisch**, das heißt, jede Aktion führt von einem Zustand in einen eindeutig bestimmten Nachfolgezustand. Es ist außerdem **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. Eine Aktion „*Fahre von München nach Ulm*“ kann – zum Beispiel durch einen Unfall – zum Nachfolgezustand „*München*“ führen. Auch kann es passieren, dass der Reisende nicht mehr weiß, wo er ist, weil er sich verirrt hat. Derartige Komplikationen wollen wir vorerst ausblenden. In diesem Kapitel werden wir daher nur Probleme betrachten, die deterministisch und beobachtbar sind.

Probleme wie das 8-Puzzle, die deterministisch und beobachtbar sind, machen die Aktionsplanung relativ einfach, denn es ist möglich, aufgrund eines abstrakten Modells Aktionssequenzen zur Lösung des Problems zu finden, ohne in der realen Welt die Aktionen auszuführen. Beim 8-Puzzle ist es nicht notwendig, tatsächlich die Plättchen in der realen Welt zu bewegen, um eine Lösung zu finden. Man kann mit so genannten **Offline-Algorithmen** optimale Lösungen finden. Ganz andere Herausforderungen hat man zum Beispiel beim Bau von Robotern, die Fußball spielen sollen. Hier wird es nie ein exaktes abstraktes Modell der Aktionen geben. Zum Beispiel kann ein Roboter, der den Ball in eine bestimmte Richtung kickt, nicht sicher vorhersagen, wohin sich der Ball bewegt, denn unter anderem weiß er nicht, ob eventuell ein Gegenspieler den Ball abfängt oder abfalscht. Hier sind dann **Online-Algorithmen** gefragt, die basierend auf Sensorsignalen in jeder Situation Entscheidungen treffen. Zur Optimierung dieser Entscheidungen basierend auf Erfahrungen dient das in Kap. 10 beschriebene **Lernen durch Verstärkung**.

6.2 Uninformierte Suche

6.2.1 Breitensuche

Bei der Breitensuche wird entsprechend dem in Abb. 6.6 angegebenen Algorithmus der Suchbaum von oben nach unten Ebene für Ebene exploriert, bis eine Lösung gefunden ist. Von links nach rechts wird für jeden Knoten in der Knotenliste zuerst getestet, ob er ein Zielknoten ist, und gegebenenfalls bei Erfolg das Programm gestoppt. Andernfalls werden alle Nachfolger des Knotens erzeugt. Auf der Liste aller neu erzeugten Knoten wird dann rekursiv die Suche fortgesetzt. Das ganze wiederholt sich, bis keine Nachfolger mehr erzeugt werden.

Dieser Algorithmus ist generisch. Das heißt, er funktioniert für beliebige Anwendungen, wenn die beiden anwendungsspezifischen Funktionen „Ziel_erreicht“ und „Nachfolger“ bereitgestellt werden. „Ziel_erreicht“ testet, ob das Argument ein Zielknoten ist, und „Nachfolger“ berechnet die Liste aller Nachfolerknoten seines Arguments. Abbildung 6.7 zeigt eine Momentaufnahme der Breitensuche.

Analyse

Da die Breitensuche jede Tiefe komplett durchsucht und jede Tiefe nach endlicher Zeit erreicht, ist sie vollständig. Die optimale, das heißt die kürzeste Lösung wird gefunden, wenn die Kosten aller Aktionen gleich sind (siehe Aufgabe 6.7). Rechenzeit und Speicherplatz wachsen exponentiell mit der Tiefe des Baumes. Für einen Baum mit konstantem Verzweigungsfaktor b und Tiefe d ergibt sich daher zusammengefasst eine Rechenzeit von

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

```

BREITENSUCHE (Knotenliste, Ziel)
NeueKnoten =  $\emptyset$ 
For all Knoten  $\in$  Knotenliste
  If Ziel_erreicht (Knoten, Ziel)
    Return („Lösung gefunden“, Knoten)
  NeueKnoten = Append (NeueKnoten, Nachfolger(Knoten))
If NeueKnoten  $\neq \emptyset$ 
  Return (BREITENSUCHE (NeueKnoten, Ziel))
Else
  Return („keine Lösung“)

```

Abb. 6.6 Der Algorithmus für die Breitensuche

Obwohl nur die letzte Ebene gespeichert wird, liegt der Speicherplatzbedarf auch bei $O(b^d)$.

Bei der Geschwindigkeit heutiger Rechner, die innerhalb von Minuten viele Milliarden von Knoten erzeugen können, wird der Hauptspeicher schnell voll sein und die Suche ist beendet. Das Problem, dass die kürzeste Lösung nicht immer gefunden wird, lässt sich lösen durch die so genannte **Uniform Cost Search**, bei der aus der aufsteigend sortierten Liste der Knoten immer die mit niedrigsten Kosten expandiert und die neuen Knoten eingesortiert werden. So findet man die optimale Lösung. Das Speicherplatzproblem ist aber noch nicht gelöst. Eine Lösung bietet die Tiefensuche.

6.2.2 Tiefensuche

Bei der Tiefensuche werden nur ganz wenige Knoten gespeichert. Nach der Expansion eines Knotens werden nur dessen Nachfolger gespeichert, und der erste Nachfolgerknoten

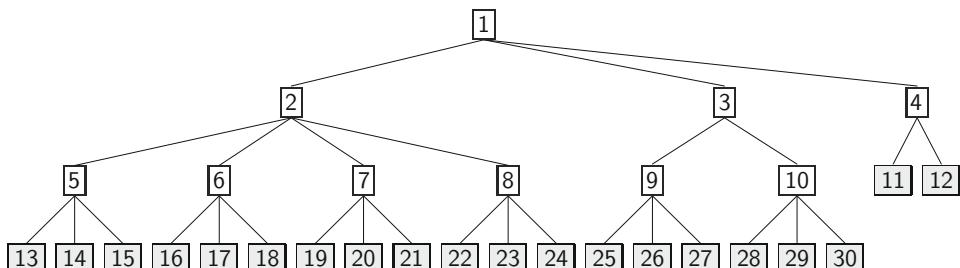


Abb. 6.7 Breitensuche während der Expansion der Knoten der dritten Ebene. Die Knoten sind nummeriert nach der Reihenfolge ihrer Erzeugung. Die Nachfolger der Knoten 11 und 12 sind noch nicht erzeugt

TIEFENSUCHE (Knoten, Ziel)

If ZielErreicht(Knoten, Ziel) Return („Lösung gefunden“)

NeueKnoten = Nachfolger (Knoten)

While NeueKnoten $\neq \emptyset$

Ergebnis = TIEFENSUCHE (Erster(NeueKnoten), Ziel)

If Ergebnis = „Lösung gefunden“ Return („Lösung gefunden“)

NeueKnoten = Rest (NeueKnoten)

Return („keine Lösung“)

Abb. 6.8 Der Algorithmus für die Tiefensuche. Die Funktion „Erster“ liefert das erste Element einer Liste und „Rest“ den Rest

wird sofort wieder expandiert. So geht die Suche sehr schnell in die Tiefe. Erst wenn ein Knoten keine Nachfolger mehr hat und die Suche in die Tiefe fehlschlägt, wird mittels **Backtracking** rückwärts bei der letzten Verzweigung der nächste offene Knoten expandiert, und so weiter. Am besten erkennt man dies an dem eleganten rekursiven Algorithmus in Abb. 6.8 und an dem Suchbaum in Abb. 6.9.

Analysis

Die Tiefensuche benötigt viel weniger Speicherplatz als die Breitensuche, denn in jeder Tiefe werden maximal b Knoten gespeichert. Man benötigt also $b \cdot d$ Speicherzellen.

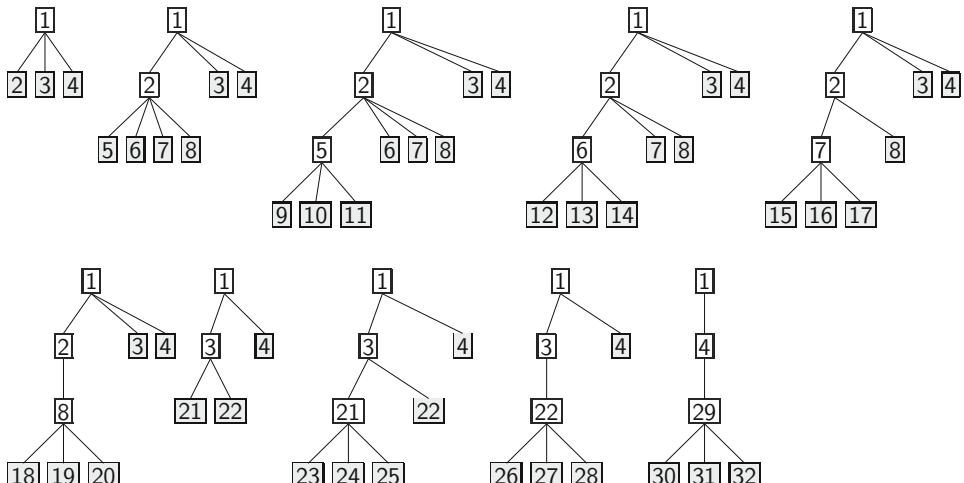


Abb. 6.9 Ablauf der Tiefensuche. Alle Knoten auf Tiefe drei sind erfolglos und führen zum Backtracking. Die Knoten sind nummeriert nach der Reihenfolge ihrer Erzeugung

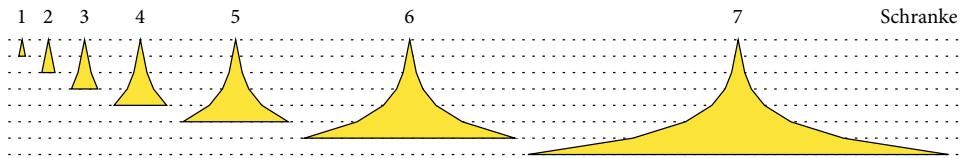


Abb. 6.10 Schematische Darstellung der Entwicklung der Suchbäume bei Iterative Deepening mit Schranken von 1 bis 7. Die Breite der Bäume entspricht einem Verzweigungsfaktor von 2

Allerdings ist die Tiefensuche nicht vollständig bei unendlich tiefen Bäumen, denn die Tiefensuche läuft in eine Endlosschleife, falls auf dem ganz linken Ast keine Lösung liegt. Damit ist natürlich auch die Frage nach dem Finden der optimalen Lösung hinfällig. Durch die Endlosschleife kann keine Schranke für die Rechenzeit angegeben werden. Im Fall eines endlich tiefen Suchbaumes mit Tiefe d werden insgesamt etwa b^d Knoten erzeugt. Also wächst die Rechenzeit, genau wie bei der Breitensuche, exponentiell mit der Tiefe.

Man kann den Suchbaum endlich machen, indem man eine **Tiefenschranke** setzt. Wenn nun in dem beschnittenen Suchbaum keine Lösung gefunden wird, kann es trotzdem noch Lösungen außerhalb der Schranke geben. Die Suche wird also nicht vollständig. Es gibt aber eine naheliegende Idee, um die Vollständigkeit der Suche zu erreichen.

6.2.3 Iterative Deepening

Man startet die Tiefensuche mit Tiefenschranke 1. Falls keine Lösung gefunden wird, erhöht man die Schranke um 1 und startet die Suche erneut, und so fort, wie in Abb. 6.10 dargestellt. Dieses iterative Erhöhen der Tiefenschranke nennt man **Iterative Deepening**.

Das in Abb. 6.8 angegebene Programm für die Tiefensuche muss man dazu durch die beiden weiteren Parameter „Tiefe“ und „Schranke“ ergänzen. „Tiefe“ wird bei dem rekursiven Aufruf um eins erhöht, und die Kopfzeile der While-Schleife wird ersetzt durch „**While** NeueKnoten $\neq \emptyset$ **Und** Tiefe < Schranke“. Die Iteration ist in Abb. 6.11 dargestellt.

Analyse

Der Speicherplatzbedarf ist gleich wie bei der Tiefensuche. Man könnte einwenden, dass durch das wiederholte Neustarten der Tiefensuche bei Tiefe null viel redundante Arbeit geleistet wird. Bei großen Verzweigungsfaktoren ist dies nicht der Fall. Wir zeigen nun, dass die Summe der Knotenzahl aller bis zur vorletzten Tiefe $d_{\max} - 1$ durchsuchten Bäume viel kleiner ist als die Zahl der Knoten im letzten durchsuchten Baum.

Sei $N_b(d)$ die Zahl der Knoten eines Suchbaumes mit Verzweigungsfaktor b und Tiefe d und d_{\max} die letzte durchsuchte Tiefe. Dann enthält der letzte durchsuchte Suchbaum

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

ITERATIVEDEEPENING (Knoten, Ziel)

Tiefenschranke = 0

Repeat

Ergebnis = TIEFENSUCHE-B (Knoten, Ziel, 0, Tiefenschranke)

Tiefenschranke = Tiefenschranke + 1

Until Ergebnis = „Lösung gefunden“

TIEFENSUCHE-B (Knoten, Ziel, Tiefe, Schranke)

If ZielErreicht (Knoten, Ziel) **Return** („Lösung gefunden“)

NeueKnoten = Nachfolger (Knoten)

While NeueKnoten $\neq \emptyset$ **Und** Tiefe < Schranke

Ergebnis =

TIEFENSUCHE-B (Erster(NeueKnoten), Ziel, Tiefe + 1, Schranke)

If Ergebnis = „Lösung gefunden“ **Return** („Lösung gefunden“)

NeueKnoten = Rest (NeueKnoten)

Return („keine Lösung“)

Abb. 6.11 Der Algorithmus für Iterative Deepening, der die leicht modifizierte Tiefensuche mit Schranke (TIEFENSUCHE-B) aufruft

Knoten. Alle davor durchsuchten Bäume zusammen besitzen

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

Knoten. Für $b > 2$ ist dies weniger als die Zahl $N_b(d_{\max})$ der Knoten im letzten Baum. Für $b = 20$ etwa enthalten die ersten $d_{\max} - 1$ Bäume zusammen nur etwa $\frac{1}{b-1} = 1/19$ der Knotenzahl des letzten Baumes. Die Rechenzeit für alle Iterationen, außer der letzten, fällt also nicht ins Gewicht.

Genau wie bei der Breitensuche ist auch hier die Vollständigkeit gegeben, und bei konstanten Kosten für alle Aktionen wird die kürzeste Lösung zuerst gefunden.

Tab. 6.1 Vergleich der uninformierten Suchalgorithmen. (*) heißtt, dass die Aussage nur bei konstanten Aktionskosten gilt. d_s ist die maximale Tiefe bei endlichem Suchbaum

	Breitensuche	Uniform cost search	Tiefensuche	Iterative deepening
Vollständigkeit	ja	ja	nein	ja
Optimale Lösung	ja (*)	ja	nein	ja (*)
Rechenzeit	b^d	b^d	∞ oder b^{d_s}	b^d
Speicherplatz	b^d	b^d	bd	bd

6.2.4 Vergleich

In Tab. 6.1 sind die vier beschriebenen Suchalgorithmen nebeneinander gestellt.

Man erkennt deutlich das Iterative Deepening als Testsieger, denn es erhält in allen Kategorien die Bestnote. Tatsächlich ist es von allen vier vorgestellten Algorithmen der einzige praktisch einsetzbare.

Wir haben nun zwar einen Testsieger, aber für realistische Anwendungen ist auch dieser meist nicht erfolgreich. Schon beim 15-Puzzle, dem größeren Bruder des 8-Puzzle (siehe Aufgabe 6.4), gibt es etwa $2 \cdot 10^{13}$ verschiedene Zustände. Bei nichttrivialen Inferenzsystemen sind die Zustandsräume noch um viele Größenordnungen größer. Wie schon in Abschn. 6.1 gezeigt, hilft hier auch alle Rechenleistung dieser Welt nicht viel weiter. Gefragt ist hingegen eine intelligente Suche, die nur einen winzigen Bruchteil des Suchraumes exploriert und darin eine Lösung findet.

6.3 Heuristische Suche

Heuristiken sind Problemlösungsstrategien, die in vielen Fällen zu einer schnelleren Lösung führen als die uninformierte Suche. Es gibt jedoch keine Garantie hierfür. Die heuristische Suche kann auch viel mehr Rechenzeit beanspruchen und letztlich dazu führen, dass die Lösung nicht gefunden wird.

Wir Menschen verwenden im Alltag mit Erfolg auf Schritt und Tritt heuristische Verfahren. Beim Gemüsekauf auf dem Markt zum Beispiel bewerten wir anhand von einigen wenigen einfachen Kriterien wie Preis, Aussehen, Herkunft und Vertrauen in den Verkäufer die verschiedenen Angebote für ein Kilo Erdbeeren und entscheiden uns dann gefühlsmäßig für das beste Angebot. Theoretisch wäre es vielleicht besser, die Erdbeeren vor dem Kauf einer gründlichen lebensmittelchemischen Analyse zu unterziehen und erst dann zu entscheiden. Zum Beispiel könnten die Erdbeeren vergiftet sein. Dann hätte sich die Analyse gelohnt. Trotzdem werden wir die Analyse nicht in Auftrag geben, denn mit hoher Wahrscheinlichkeit ist unsere heuristische Auswahl erfolgreich und führt schnell zu dem angestrebten Ziel, schmackhafte Erdbeeren essen zu können.

Heuristische Entscheidungen sind eng verknüpft mit der Notwendigkeit, **Realzeitentscheidungen unter beschränkten Ressourcen** herbeizuführen. In der Praxis wird eine

```

HEURISTISCHESUCHE (Start, Ziel)
Knotenliste = [Start]
While True
  If Knotenliste =  $\emptyset$  Return („keine Lösung“)
  Knoten = Erster (Knotenliste)
  Knotenliste = Rest (Knotenliste)
  If Ziel_erreicht (Knoten, Ziel) Return („Lösung gefunden“, Knoten)
  Knotenliste = Einsortieren (Nachfolger(Knoten),Knotenliste)

```

Abb. 6.12 Der Algorithmus für die heuristische Suche

schnell gefundene gute Lösung einer optimalen, aber nur mit großem Aufwand herbeigeführten, Entscheidung vorgezogen.

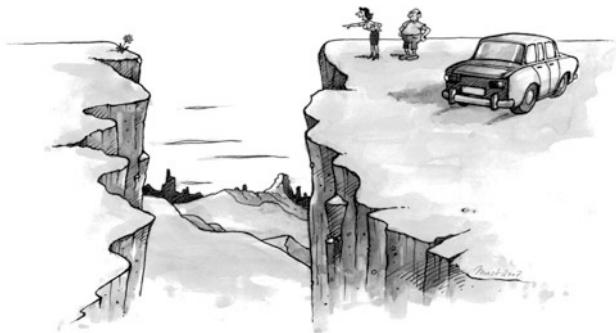
Zur mathematischen Modellierung einer Heuristik wird eine **heuristische Bewertungsfunktion** $f(s)$ für Zustände verwendet. Ziel ist es, mit Hilfe dieser Bewertung mit wenig Aufwand eine Lösung der gestellten Suchaufgabe mit minimalen Gesamtkosten zu finden. Hierbei ist es wichtig, zu unterscheiden zwischen dem Aufwand zum Finden einer Lösung und den Gesamtkosten dieser Lösung. Zum Beispiel findet Google Maps eine Route von Ravensburg nach Kathmandu in 2 Sekunden, aber die Fahrt von Ravensburg nach Kathmandu dauert mindestens 109 Stunden reine Fahrzeit. Außerdem fallen noch andere Kosten zum Beispiel für Benzin an (Gesamtkosten).

Zunächst werden wir den Algorithmus für die Breitensuche modifizieren und mit einer heuristischen Bewertungsfunktion versehen. Die aktuellen offenen Knoten auf der letzten Ebene werden nun nicht mehr von links nach rechts der Reihe nach expandiert, sondern entsprechend ihren heuristischen Bewertungen. Aus der Menge der offenen Knoten wird immer zuerst ein Knoten mit minimaler Bewertung expandiert. Dies wird dadurch erreicht, dass alle expandierten Knoten sofort bewertet und dann entsprechend ihrer Bewertung in die Liste der offenen Knoten eingesortiert werden. Die Liste kann nun Knoten aus unterschiedlichen Tiefen des Suchbaumes enthalten.

Da die heuristische Bewertung von Zuständen für die Suche sehr wichtig ist, werden wir im Folgenden zwischen dem **Zustand** und dem zugehörigen **Knoten** unterscheiden. Der Knoten enthält den Zustand und noch weitere für die Suche relevante Informationen wie zum Beispiel die Tiefe im Suchbaum und die heuristische Bewertung des Zustands. Demzufolge muss die Funktion „Nachfolger“, welche die Nachfolger eines Knotens erzeugt, auch für eben diese Nachfolgezustände sofort deren heuristische Bewertung als Bestandteil des Knotens berechnen. Wir definieren dazu den allgemeinen Suchalgorithmus **HEURISTISCHESUCHE** in Abb. 6.12.

Die Knotenliste wird initialisiert mit dem Startknoten. Dann wird in der Schleife der erste Knoten aus der Liste entfernt und getestet, ob er ein Lösungsknoten ist. Wenn nicht, wird er durch die Funktion „Nachfolger“ expandiert und seine Nachfolger mit Hilfe der Funktion „Einsortieren“ in die Liste eingefügt. „Einsortieren(X,Y)“ fügt die Elemente aus

Abb. 6.13 Er: „Schatz, denk mal an die Spritkosten! Ich pflück dir woanders welche.“ Sie: „Nein, ich will die da!“



der unsortierten Liste X in die aufsteigend sortierte Liste Y ein. Als Sortierschlüssel wird die heuristische Bewertung verwendet. So wird sichergestellt, dass immer die *besten* Knoten – das heißt die Knoten mit der niedrigsten heuristischen Bewertung – am Anfang der Liste stehen.³

Tiefensuche und Breitensuche sind übrigens Spezialfälle der Funktion HEURISTISCHE-SUCHE. Durch Austauschen der Bewertungsfunktion für die Knoten lassen sie sich einfach erzeugen (Aufgabe 6.11).

Die beste Heuristik wäre eine Funktion, die für jeden Knoten die tatsächlichen Kosten zum Ziel berechnet. Dazu müsste aber der ganze Suchraum durchsucht werden, was durch die Heuristik gerade verhindert werden soll. Wir brauchen also eine Heuristik, die schnell und einfach zu berechnen ist. Wie findet man solch eine Heuristik?

Eine interessante Idee zum Finden einer Heuristik ist die Vereinfachung der Aufgabenstellung. Dabei wird die ursprüngliche Aufgabe soweit vereinfacht, dass sie mit wenig Rechenaufwand zu lösen ist. Während der Suche wird nun für jeden Zustand das vereinfachte Problem gelöst. Die Kosten vom Zustand zum Ziel für das vereinfachte Problem dienen dann als Abschätzung der Kosten für das eigentliche Problem (siehe Abb. 6.13). Diese **Kostenschätzfunktion** wird mit h bezeichnet.

6.3.1 Gierige Suche

Es liegt nahe, unter den aktuell zur Auswahl stehenden Zuständen den mit dem kleinsten h -Wert, das heißt mit den niedrigsten geschätzten Kosten, auszuwählen. Die Kostenschätzfunktion h verwenden wir nun also direkt als heuristische Bewertungsfunktion. Wir setzen für die Bewertung neuer Knoten in der Funktion HEURISTISCHESUCHE $f(s) = h(s)$. Am Beispiel der Routenplanung (Beispiel 6.2) lässt sich dies gut veranschaulichen. Als vereinfachtes Problem stellen wir uns die Aufgabe, auf dem geraden Weg, das heißt auf der Luftlinie, vom Start zum Ziel zu kommen. Statt der optimalen Route suchen wir nun al-

³ Während des Einsortierens eines neuen Knotens in die Knotenliste kann es eventuell vorteilhaft sein, zu prüfen, ob der neue Knoten schon vorhanden ist und gegebenenfalls das Duplikat zu löschen.

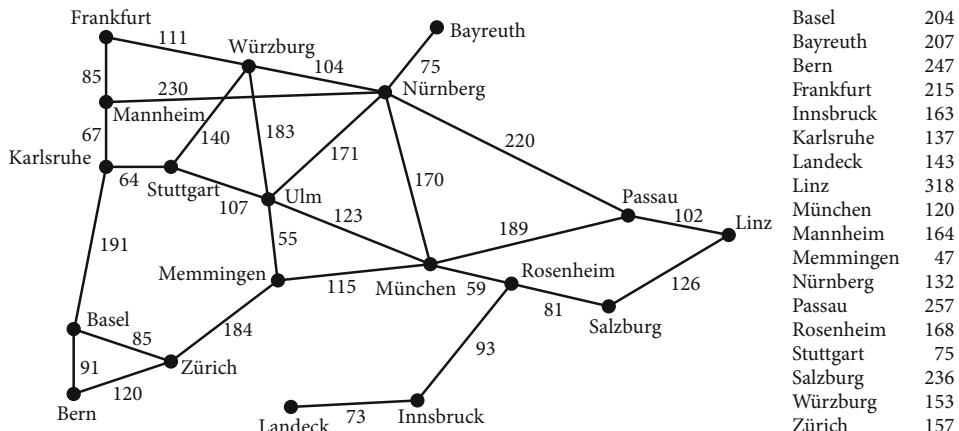


Abb. 6.14 Städtegraph mit Luftlinienentfernungen aller Städte nach Ulm

so zuerst von jedem Knoten eine Route mit minimaler Luftliniendistanz zum Ziel. Als Ziel wählen wir Ulm. Damit ergibt sich als Kostenschätzfunktion

$h(s)$ = Luftlinienentfernung von Stadt s nach Ulm.

Die Luftlinienentferungen von allen Städten nach Ulm sind in Abb. 6.14 neben dem Graphen angegeben.

In Abb. 6.15 links ist der durch HEURISTISCHESUCHE explorierte Suchbaum für Start in Linz dargestellt. Man erkennt gut, dass der Suchbaum sehr schlank ist. Die Suche kommt also sehr schnell zum Ziel. Leider findet sie nicht immer die optimale Lösung. Zum Beispiel bei Start in Mannheim findet dieser Algorithmus nicht die optimale Lösung (Abb. 6.15 rechts). Der gefundene Weg Mannheim–Nürnberg–Ulm hat eine Länge von 401 km. Die Route Mannheim–Karlsruhe–Stuttgart–Ulm wäre mit 238 km aber deutlich kürzer. Be- trachtet man den Graphen, so wird die Ursache des Problems deutlich. Nürnberg liegt zwar etwas näher an Ulm als Karlsruhe, aber die Entfernung von Mannheim nach Nürnberg ist deutlich größer als die von Mannheim nach Karlsruhe. Die Heuristik schaut nur „gierig“ nach vorne zum Ziel, anstatt auch die bis zum aktuellen Knoten schon zurückgelegte Strecke zu berücksichtigen. Dies ist der Grund für den Namen **gierige Suche** (engl. **greedy search**).

6.3.2 A^{*}-Suche

Wir wollen nun die auf der Suche bis zum aktuellen Knoten s angefallenen Kosten mit berücksichtigen. Zuerst definieren wir die **Kostenfunktion**

$g(s)$ = Summe der vom Start bis zum aktuellen Knoten angefallenen Kosten,

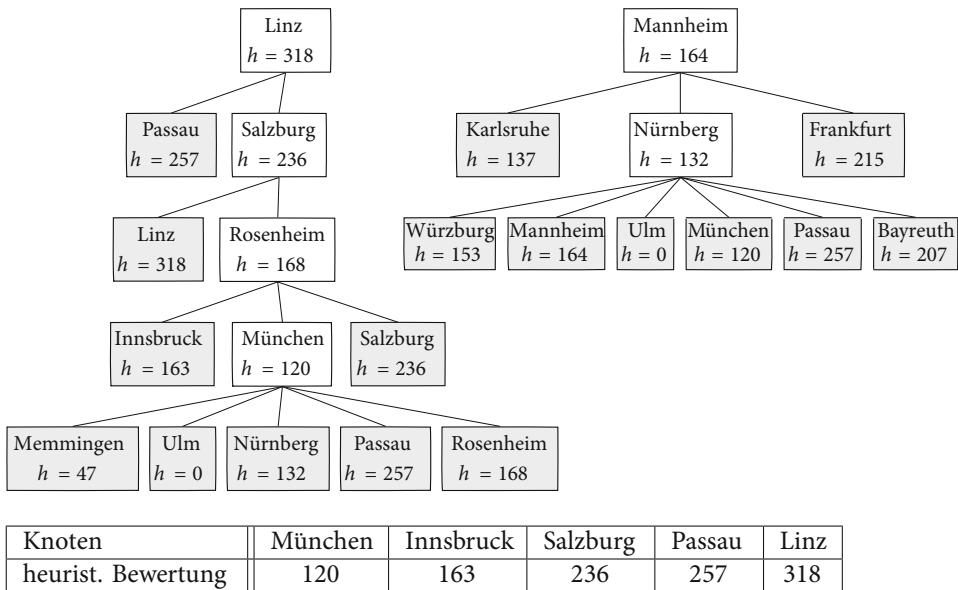


Abb. 6.15 Greedy-Suche: von Linz nach Ulm (*links*) und von Mannheim nach Ulm (*rechts*). Für den linken Suchbaum ist unten die nach Bewertung sortierte Datenstruktur Knotenliste vor der Expansion des Knotens München angegeben

addieren die geschätzten Kosten bis zum Ziel dazu und erhalten als **heuristische Bewertungsfunktion**

$$f(s) = g(s) + h(s).$$

Nun stellen wir außerdem noch eine kleine aber wichtige Forderung.

Definition 6.4

Eine heuristische Kostenschätzfunktion $h(s)$, welche die tatsächlichen Kosten vom Zustand s zum Ziel nie überschätzt, heißt **zulässig (engl. admissible)**.

Die Funktion HEURISTISCHESUCHE zusammen mit einer Bewertungsfunktion $f(s) = g(s) + h(s)$ und zulässiger Heuristik h wird als **A^{*}-Algorithmus** bezeichnet. Dieser berühmte Algorithmus ist vollständig und optimal. A^{*} findet also bei jedem lösbar Problem immer die kürzeste Lösung, was wir im Folgenden verstehen und beweisen wollen.

Zuerst wenden wir den A^{*}-Algorithmus auf das Beispiel an. Gesucht ist die kürzeste Verbindung von Frankfurt nach Ulm.

Im oberen Teil von Abb. 6.16 erkennt man, dass die Nachfolger von Mannheim auf Grund des besseren f -Wertes vor den Nachfolgern von Würzburg erzeugt werden. Die

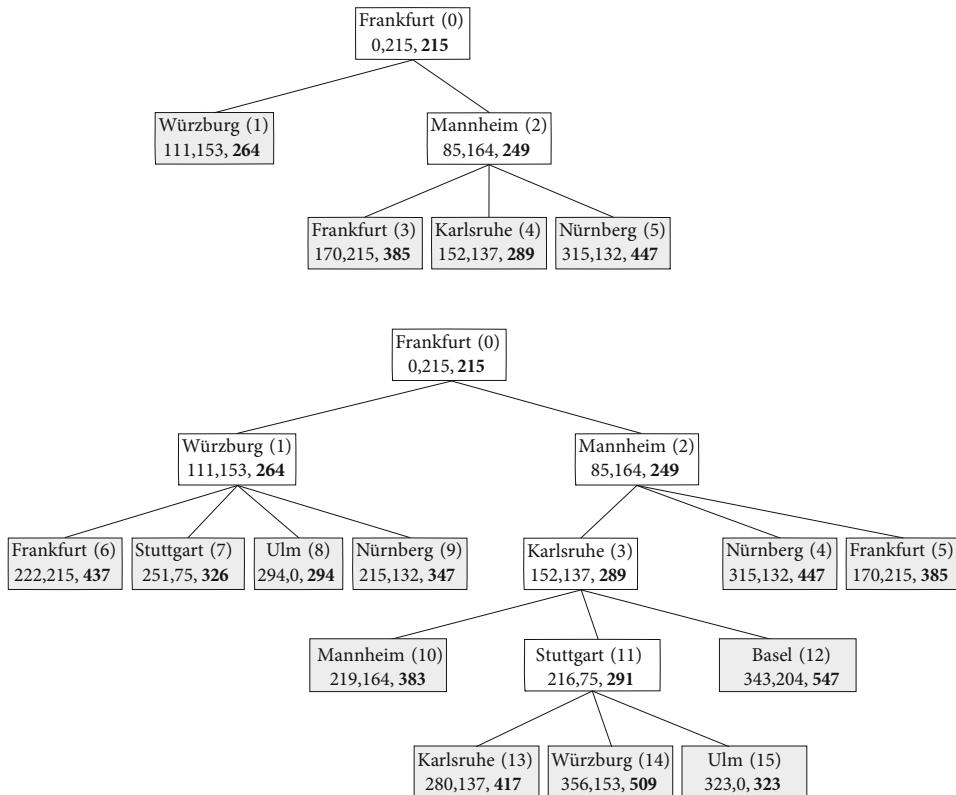


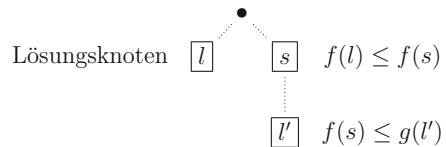
Abb. 6.16 Zwei Momentaufnahmen des Suchbaums der A^* -Suche für eine optimale Route von Frankfurt nach Ulm. Unter dem Namen der Stadt s ist jeweils $g(s)$, $h(s)$, $f(s)$ angegeben. Nummern in Klammer neben einem Städtenamen geben die Reihenfolge der Erzeugung des Knotens durch die Funktion „Nachfolger“ an

optimale Lösung Frankfurt–Würzburg–Ulm wird kurz darauf im achten Schritt erzeugt, aber noch nicht als solche erkannt. Daher terminiert der Algorithmus noch nicht, denn der Knoten Karlsruhe (3) hat einen besseren f -Wert und ist daher vor dem Knoten Ulm (8) an der Reihe. Erst wenn alle f -Werte größer oder gleich dem des Lösungsknotens Ulm (8) sind, ist sichergestellt, dass es sich hier um eine optimale Lösung handelt. Andernfalls könnte eventuell noch eine Lösung mit geringeren Kosten gefunden werden. Dies wollen wir nun allgemein zeigen.

Satz 6.2

Der A^* -Algorithmus ist optimal. Das heißt, er findet immer die Lösung mit den niedrigsten Gesamtkosten, wenn die Heuristik h zulässig ist.

Abb. 6.17 Der von A* zuerst gefundene Lösungsknoten l hat nie höhere Kosten als ein beliebiger anderer Lösungsknoten l'



Beweis Im Algorithmus HEURISTISCHESUCHE wird jeder neu erzeugte Knoten s durch die Funktion „Einsortieren“ immer entsprechend seiner heuristischen Bewertung $f(s)$ eingesortiert. Der Knoten mit der kleinsten Bewertung steht also am Anfang der Liste. Ist nun der Knoten l am Anfang der Liste ein Lösungsknoten, so hat kein anderer Knoten eine bessere heuristische Bewertung. Für alle anderen Knoten s gilt daher $f(l) \leq f(s)$. Da die Heuristik zulässig ist, kann auch nach Expansion aller anderen Knoten keine bessere Lösung l' gefunden werden (siehe Abb. 6.17). Formal aufgeschrieben heißt das

$$g(l) = g(l) + h(l) = f(l) \leq f(s) = g(s) + h(s) \leq g(l').$$

Die erste Gleichung gilt, weil l ein Lösungsknoten ist mit $h(l) = 0$. Die zweite ist die Definition von f . Die dritte (Un-)Gleichung gilt, weil die Liste der offenen Knoten aufsteigend sortiert ist. Die vierte Gleichung ist wieder die Definition von f . Die letzte (Un-)Gleichung schließlich ist die Zulässigkeit der Heuristik, die die Kosten vom Knoten s zu einer beliebigen Lösung nicht überschätzt. Damit ist gezeigt, dass $g(l) \leq g(l')$, das heißt, dass die gefundene Lösung l optimal ist. \square

6.3.3 IDA*-Suche

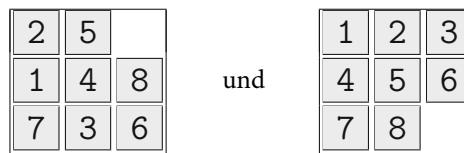
Die Suche mit A* hat noch einen Makel von der Breitensuche geerbt. Es werden immer noch viele Knoten gespeichert, was zu hohem Speicherplatzbedarf führen kann. Außerdem muss die Liste der offenen Knoten sortiert sein. Damit lässt sich das Einfügen neuer Knoten in die Liste und das Entnehmen von Knoten aus der Liste nicht mehr in konstanter Zeit durchführen, was die Komplexität des Algorithmus leicht erhöht. Basierend auf dem Heapsort-Sortieralgorithmus kann man die Knotenliste als Heap strukturieren mit logarithmischer Zeitkomplexität für das Einfügen und Entnehmen von Knoten (siehe [CLR90]).

Beide Probleme lassen sich – analog wie bei der Breitensuche – lösen durch Iterative Deepening. Man arbeitet mit Tiefensuche und erhöht die Schranke sukzessive. Allerdings wird hier nicht mit einer Tiefenschranke gearbeitet, sondern mit einer Schranke für die heuristische Bewertung $f(s)$. Dieses Verfahren wird als IDA*-Algorithmus bezeichnet.

6.3.4 Empirischer Vergleich der Suchalgorithmen

Mit A* beziehungsweise IDA* haben wir einen Suchalgorithmus mit vielen guten Eigenschaften. Er ist vollständig und optimal. Er kann also risikolos benutzt werden. Das wichtigste ist aber, dass er mit Heuristiken arbeitet und daher in vielen Fällen die Rechenzeiten bis zum Finden einer Lösung deutlich reduziert. Dies wollen wir am Beispiel des 8-Puzzle nun empirisch untersuchen.

Für das 8-Puzzle gibt es zwei einfache zulässige Heuristiken. Die Heuristik h_1 zählt einfach die Anzahl der Plättchen, die nicht an der richtigen Stelle liegen. Offensichtlich ist diese Heuristik zulässig. Heuristik h_2 misst den **Manhattan-Abstand**. Für jedes Plättchen werden horizontaler und vertikaler Abstand zum gleichen Plättchen im Zielzustand addiert. Dieser Wert wird dann über alle Plättchen aufsummiert, was den Wert $h_2(s)$ ergibt. Zum Beispiel berechnet sich der Manhattan-Abstand der beiden Zustände



zu

$$h_2(s) = 1 + 1 + 1 + 1 + 2 + 0 + 3 + 1 = 10.$$

Auch beim Manhattan-Abstand ist die Zulässigkeit der Heuristik offensichtlich (siehe Aufgabe 6.13).

Die beschriebenen Algorithmen wurden mit Mathematica implementiert. Zum Vergleich mit der uninformierten Suche wurden der A*-Algorithmus mit den beiden Heuristiken h_1 und h_2 sowie Iterative Deepening auf 132 zufällig generierte lösbare 8-Puzzle-Probleme angewendet. Die Mittelwerte der Zahl der Schritte und der Rechenzeiten sind in Tab. 6.2 angegeben. Man erkennt, dass die Heuristiken den Suchaufwand gegenüber der uninformierten Suche deutlich reduzieren.

Vergleicht man zum Beispiel bei Tiefe 12 Iterative Deepening mit A*(h_1), so zeigt sich, dass h_1 die Zahl der Schritte um etwa den Faktor 3000 reduziert, die Rechenzeit aber nur um den Faktor 1023. Dies liegt an dem höheren Aufwand pro Schritt für die Berechnung der Heuristik.

Bei genauem Hinsehen fällt in den Spalten für h_1 und h_2 ein Sprung in der Zahl der Schritte von Tiefe 12 nach Tiefe 14 auf. Dieser Sprung kann nicht allein der wiederholten Arbeit durch IDA* zugeschrieben werden. Er kommt daher, dass die hier verwendete Implementierung des A*-Algorithmus Duplikate von identischen Knoten löscht und dadurch den Suchraum verkleinert. Bei IDA* ist dies nicht möglich, da fast keine Knoten gespeichert werden. Trotzdem ist A* ab Tiefe 14 nicht mehr konkurrenzfähig mit IDA*, denn durch den Aufwand für das Einfügen neuer Knoten steigt die Zeit pro Schritt stark an.

Tab. 6.2 Vergleich des Rechenaufwands von uninformerter Suche und heuristischer Suche für lösbare 8-Puzzle-Probleme mit verschiedenen Tiefen. Messung in Schritten und Sekunden. Alle Werte sind Mittelwerte über mehrere Läufe (siehe letzte Spalte)

Tiefe	Iterative Deepening		A [*] -Algorithmus							
	Schritte	Zeit [s]	Heuristik h_1		Heuristik h_2		Schritte	Zeit [s]		
			Schritte	Zeit [s]	Schritte	Zeit [s]				
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	50.512	7,9	48,1	0,018	22,1	0,011	15			
12	486.751	75,7	162,2	0,074	56,0	0,031	12			
IDA [*]										
14	–	–	10.079,2	2,6	855,6	0,25	16			
16	–	–	69.386,6	19,0	3806,5	1,3	13			
18	–	–	708.780,0	161,6	53.941,5	14,1	4			

Eine Berechnung des effektiven Verzweigungsfaktors nach (6.1) ergibt Werte von etwa 2,8 für die uninformerter Suche. Dieser Wert stimmt überein mit dem Wert aus Abschn. 6.1. Die Heuristik h_1 reduziert den Verzweigungsfaktor auf Werte um etwa 1,5 und h_2 auf etwa 1,3. An der Tabelle erkennt man gut, dass eine kleine Reduktion des Verzweigungsfaktors von 1,5 auf 1,3 einen großen Gewinn an Rechenzeit bringt.

Die heuristische Suche hat damit eine wichtige Bedeutung für die Praxis, denn sie erlaubt die Lösung von Problemen, die weit außerhalb der Reichweite uninformerter Suche liegen.

6.3.5 Zusammenfassung

Von den verschiedenen Suchalgorithmen für uninformerter Suche ist Iterative Deepening der einzige praktisch einsetzbare, denn er ist vollständig und kommt mit sehr wenig Speicher aus. Bei schwierigen kombinatorischen Suchproblemen scheitert jedoch auch das Iterative Deepening meist an der Größe des Suchraumes. Oft hilft hier die heuristische Suche durch ihre Reduktion des effektiven Verzweigungsfaktors. Mit dem IDA^{*}-Algorithmus existiert heute ein Verfahren, das wie das Iterative Deepening vollständig ist und nur sehr wenig Speicher benötigt.

Heuristiken bringen natürlich nur dann einen deutlichen Gewinn, wenn die Heuristik „gut“ ist. Die eigentliche Aufgabe des Entwicklers beim Lösen schwieriger Suchprobleme besteht daher im Entwurf von Heuristiken, die den effektiven Verzweigungsfaktor stark verkleinern. In Abschn. 6.5 werden wir uns mit dieser Problematik befassen und auch aufzeigen, wie maschinelle Lernverfahren zum automatischen Generieren von Heuristiken verwendet werden können.

Abschließend bleibt noch anzumerken, dass Heuristiken bei unlösbaren Problemen keinen Gewinn bringen, denn die Unlösbarkeit eines Problems kann erst dann festgestellt werden, wenn der komplette Suchbaum durchsucht wurde. Bei entscheidbaren Problemen wie etwa dem 8-Puzzle bedeutet dies, dass bis zu einer maximalen Tiefe der ganze Suchraum durchsucht werden muss, egal ob nun eine Heuristik verwendet wird oder nicht. Die Heuristik bedeutet in diesem Fall immer einen Nachteil, bedingt durch den höheren Rechenaufwand für das Auswerten der Heuristik. Dieser Nachteil lässt sich aber meist unabhängig von der Problemgröße durch einen konstanten Faktor abschätzen. Bei unentscheidbaren Problemen wie zum Beispiel beim Beweisen von PL1-Formeln kann der Suchbaum unendlich tief sein. Das heißt, die Suche terminiert im nicht lösbarer Fall eventuell gar nicht. Zusammenfassend lässt sich folgendes aussagen: Für lösbare Probleme reduzieren Heuristiken die Rechenzeiten oft dramatisch, für unlösbare Probleme hingegen kann der Aufwand mit Heuristik sogar ansteigen.

6.4 Spiele mit Gegner

Spiele für zwei Spieler wie zum Beispiel Schach, Dame, Reversi oder Go sind deterministisch, denn jede Aktion (ein Spielzug) führt bei gleichem Ausgangszustand immer zum gleichen Nachfolgezustand. Im Unterschied dazu ist Backgammon nichtdeterministisch, denn hier hängt der Nachfolgezustand vom Würfelergebnis ab. Diese Spiele sind alle beobachtbar, denn jeder Spieler kennt immer den kompletten Spielzustand. Viele Kartenspiele, wie zum Beispiel Skat, sind nur teilweise beobachtbar, denn der Spieler kennt die Karten des Gegners nicht oder nur teilweise.

Die bisher in diesem Kapitel behandelten Problemstellungen waren deterministisch und beobachtbar. Im Folgenden werden wir auch nur Spiele betrachten, die deterministisch und beobachtbar sind. Außerdem beschränken wir uns auf **Nullsummenspiele**. Dies sind Spiele, bei denen jeder Gewinn eines Spielers einen Verlust des Gegenspielers in gleicher Höhe bedeutet. Die Summe aus Gewinn und Verlust ist also immer gleich null. Dies trifft auf die oben erwähnten Spiele Schach, Dame, Reversi und Go zu.

6.4.1 Minimax-Suche

Ziel jedes Spielers ist es, optimale Züge zu machen, die zum Sieg führen. Im Prinzip wäre es denkbar, ähnlich wie beim 8-Puzzle einen Suchbaum aufzubauen und komplett zu durchsuchen nach einer Folge von Zügen, die zum Sieg führt. Allerdings sind hier einige Besonderheiten zu beachten:

- Der effektive Verzweigungsfaktor beim Schachspiel liegt etwa bei 30 bis 35. Bei einem typischen Spiel mit 50 Zügen pro Spieler hat der Suchbaum dann mehr als $30^{100} \approx 10^{148}$ Blattknoten. Der Suchbaum lässt sich also bei weitem nicht vollständig explorieren. Hinzu

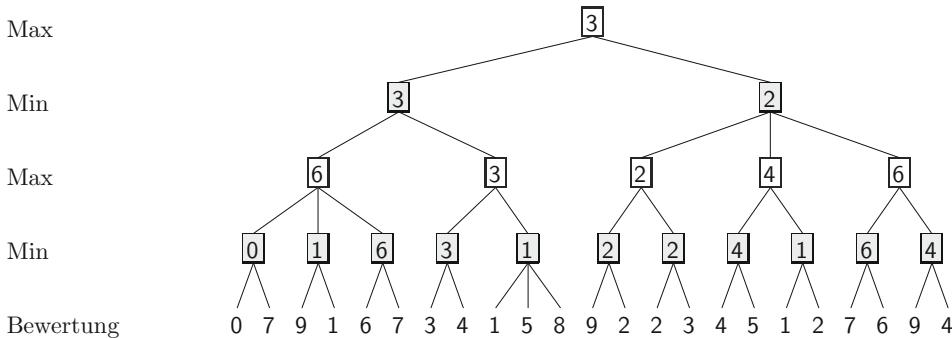


Abb. 6.18 Ein Minimax-Spielbaum mit Vorausschau von 4 Halbzügen

kommt, dass beim Schachspiel oft mit Zeitbeschränkung gespielt wird. Wegen dieser **Realzeitanforderungen** wird die Tiefe des Suchbaumes auf eine passende Tiefe, zum Beispiel acht Halbzüge, beschränkt. Da unter den Blattknoten dieses stark beschränkten Baumes meist keine Lösungsknoten sind, verwendet man eine **Bewertungsfunktion** B zur heuristischen Bewertung von Stellungen. Die Spielstärke des Programms hängt stark von der Güte der Bewertungsfunktion ab. Daher werden wir in Abschn. 6.5 dieses Thema näher behandeln.

2. Im Folgenden bezeichnen wir den Spieler, dessen Spielweise wir optimieren wollen, mit Max und seinen Gegner mit Min. Die Züge des Gegners (Min) sind im Voraus nicht bekannt, und damit auch der tatsächliche Suchbaum nicht. Dieses Problem kann man elegant lösen, indem man annimmt, dass der Gegner immer den für ihn besten Zug macht. Je höher für eine Stellung s die Bewertung $B(s)$, desto besser ist die Stellung für den Spieler Max und desto schlechter ist sie für seinen Gegner Min. Max versucht also, die Bewertung seiner Züge zu maximieren, wohingegen Min Züge macht, die eine möglichst kleine Bewertung erhalten. Wenn nun Max seinen Gegner Min nicht kennt, sollte er, um zu gewinnen, immer davon ausgehen, dass Min den aus seiner Sicht optimalen Zug macht.

In Abb. 6.18 ist ein Suchbaum mit vier Halbzügen und Bewertungen aller Blätter angegeben. Die Bewertung eines inneren Knotens ergibt sich rekursiv, je nach Ebene als Minimum oder Maximum der Werte seiner Nachfolerknoten.

6.4.2 Alpha-Beta-Pruning

Durch den Wechsel zwischen Maximierung und Minimierung kann man sich unter Umständen viel Arbeit sparen. Das so genannte Alpha-Beta-Pruning arbeitet mit Tiefensuche bis zur eingestellten Tiefenschanke. So wird nun der Spielbaum von links nach rechts durchsucht. Wie bei der Minimax-Suche wird an den Minimumknoten das Minimum aus den Werten der Nachfolger gebildet und an den Maximumknoten das Maximum.

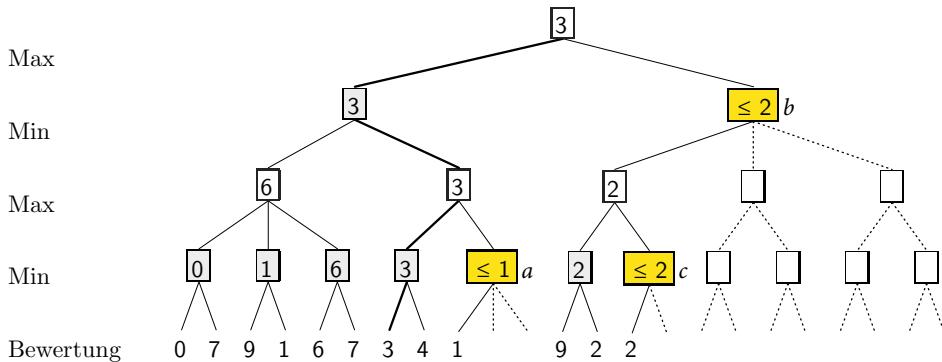


Abb. 6.19 Ein Alphabeta-Spielbaum mit Vorausschau von 4 Halbzügen. Die gestrichelten Teile des Suchbaumes werden nicht traversiert, denn sie haben keinen Einfluss auf das Ergebnis

In Abb. 6.19 ist für den Baum aus Abb. 6.18 dieses Verfahren dargestellt. An dem mit *a* markierten Knoten können nach Bewertung des ersten Nachfolgers mit dem Wert 1 alle anderen Nachfolger ignoriert werden, denn das Minimum ist sicher ≤ 1 . Es könnte zwar noch kleiner werden, aber das ist irrelevant, da eine Ebene höher das Maximum schon ≥ 3 ist. Egal, wie die Bewertung der restlichen Nachfolger von *a* ausfällt, das Maximum bleibt beim Wert 3. Analog wird beim Knoten *b* abgeschnitten. Da der erste Nachfolger von *b* den Wert 2 hat, kann das in *b* zu bildende Minimum nur kleiner oder gleich 2 sein. Das Maximum am Wurzelknoten ist aber schon sicher ≥ 3 . Dieses wird durch Werte ≤ 2 nicht verändert. Also können die restlichen Unteräste von *b* abgeschnitten werden.

Die gleiche Argumentation gilt auch für den Knoten *c*. Allerdings ist hier der relevante Maximumknoten nicht der direkte Vorgänger, sondern der Wurzelknoten. Dies lässt sich verallgemeinern.

- An jedem Blattknoten wird die Bewertung berechnet.
- Für jeden Maximumknoten wird während der Suche der aktuell größte Wert der bisher traversierten Nachfolger in α gespeichert.
- Für jeden Minimumknoten wird während der Suche der aktuell kleinste Wert der bisher traversierten Nachfolger in β gespeichert.
- Ist an einem Minimumknoten *k* der aktuelle Wert $\beta \leq \alpha$, so kann die Suche unter *k* beendet werden. Hierbei ist α der größte Wert eines Maximumknotens im Pfad von der Wurzel zu *k*.
- Ist an einem Maximumknoten *l* der aktuelle Wert $\alpha \geq \beta$, so kann die Suche unter *l* beendet werden. Hierbei ist β der kleinste Wert eines Minimumknotens im Pfad von der Wurzel zu *l*.

Der in Abb. 6.20 angegebene Algorithmus ist eine Erweiterung der Tiefensuche mit zwei sich abwechselnd aufrufenden Funktionen für die Maximum- und Minimumknoten. Er verwendet die oben definierten Werte α und β .

```

ALPHABETAMAX (Knoten,  $\alpha$ ,  $\beta$ )
If TiefenschrankeErreicht (Knoten) Return (Bewertung(Knoten))
NeueKnoten = Nachfolger (Knoten)
While NeueKnoten  $\neq \emptyset$ 
     $\alpha$  = Maximum ( $\alpha$ , ALPHABETAMIN (Erster(NeueKnoten),  $\alpha$ ,  $\beta$ ))
    If  $\alpha \geq \beta$  Return ( $\beta$ )
    NeueKnoten = Rest (NeueKnoten)
Return( $\alpha$ )

```

```

ALPHABETAMIN (Knoten,  $\alpha$ ,  $\beta$ )
If TiefenschrankeErreicht (Knoten) Return (Bewertung(Knoten))
NeueKnoten = Nachfolger (Knoten)
While NeueKnoten  $\neq \emptyset$ 
     $\beta$  = Minimum ( $\beta$ , ALPHABETAMAX (Erster(NeueKnoten),  $\alpha$ ,  $\beta$ ))
    If  $\beta \leq \alpha$  Return ( $\alpha$ )
    NeueKnoten = Rest (NeueKnoten)
Return ( $\beta$ )

```

Abb. 6.20 Der Algorithmus für die Alpha-Beta-Suche mit den zwei Funktionen ALPHABETAMIN und ALPHABETAMAX

Komplexität

Die Rechenzeitersparnis des Alpha-Beta-Pruning gegenüber der einfachen Minimax-Suche hängt stark von der Reihenfolge ab in der die Nachfolgerknoten traversiert werden. Im Worst-Case bringt das Alpha-Beta-Pruning keinen Gewinn. Dann ist bei festem Verzweigungsfaktor b die Zahl n_d der zu bewertenden Blattknoten auf Tiefe d gleich

$$n_d = b^d.$$

Im Best-Case, wenn die Nachfolger von Maximumknoten absteigend sortiert sind und die Nachfolger von Minimumknoten aufsteigend sind, reduziert sich der effektive Verzweigungsfaktor auf \sqrt{b} . Bei Schach bedeutet dies eine erhebliche Reduktion des effektiven Verzweigungsfaktors von 35 auf etwa 6. Es werden dann also noch

$$n_d = \sqrt{b}^d = b^{d/2}$$

Blattknoten erzeugt. Das bedeutet, dass sich die Tiefenschranke und damit der Suchhorizont mit Alpha-Beta-Pruning verdoppelt. Allerdings gilt dies nur für den Fall optimal sortierter Nachfolger, denn zum Zeitpunkt der Erzeugung der Nachfolgerknoten ist deren Bewertung noch nicht bekannt. Sind die Nachfolger zufällig sortiert, so reduziert sich der

Verzweigungsfaktor auf $b^{3/4}$ und die Zahl der Blattknoten auf

$$n_d = b^{\frac{3}{4}d}.$$

Bei gleicher Rechenleistung kann also zum Beispiel ein Schachcomputer mit Alpha-Beta-Pruning acht statt nur sechs Halbzüge vorausberechnen, bei einem effektiven Verzweigungsfaktor von etwa 14. Eine fundierte Analyse mit Herleitung dieser Schranken ist zu finden in [Pea84].

Um, wie oben erwähnt, bei einem Schachcomputer die Suchtiefe zu verdoppeln, benötigt man eine optimale Ordnung der Nachfolger, die man aber in der Praxis nicht hat. Sonst wäre die Suche unnötig. Mit einem einfachen Trick erhält man eine relativ gute Knotenordnung. Man verbindet das Alpha-Beta-Pruning mit Iterative Deepening über die Tiefenschranke. So kann man bei jeder neuen Tiefenschranke auf die Bewertungen aller Knoten der vorangegangenen Ebenen zugreifen und an jeder Verzweigung die Nachfolger anordnen. Dadurch erreicht man einen effektiven Verzweigungsfaktor von etwa 7 bis 8, was nicht mehr weit von dem theoretischen Optimum $\sqrt{35}$ entfernt ist [Nil98].

6.4.3 Nichtdeterministische Spiele

Die Minimax-Suche lässt sich auf Spiele mit nichtdeterministischen Aktionen, zum Beispiel Backgammon, verallgemeinern. Jeder Spieler würfelt vor seinem Zug, der vom Ergebnis des Wurfes mit beeinflusst wird. Im Spielbaum gibt es nun also drei Arten von Ebenen in der Reihenfolge

Max, Würfeln, Min, Würfeln, ...,

wobei jeder Würfelknoten sechsfach verzweigt. Da man die Augenzahl des Würfels nicht vorhersagen kann, mittelt man die Bewertungen aller Würfe und führt dann die Suche wie beschrieben mit den Mittelwerten aus [RN03].

6.5 Heuristische Bewertungsfunktionen

Wie findet man für eine Suchaufgabe eine gute heuristische Bewertungsfunktion? Hier gibt es grundsätzlich zwei Ansätze. Der klassische Weg verwendet das Wissen menschlicher Experten. Der KI-Entwickler (engl. knowledge engineer) hat nun die meist sehr schwierige Aufgabe, das implizite Wissen des Experten in Form eines Computerprogramms zu formalisieren. Am Beispiel des Schachspiels wollen wir nun aufzeigen, wie dieser Prozess vereinfacht werden kann.

Im ersten Schritt wird der Experte nur nach den wichtigsten Größen für die Auswahl eines Zuges befragt. Dann wird versucht, diese Größen numerisch zu quantifizieren. Man

erhält eine Liste relevanter Features oder auch Attribute. Diese werden nun (im einfachsten Fall) zu einer linearen Bewertungsfunktion $B(s)$ für Stellungen kombiniert, die etwa so aussehen könnte

$$B(s) = a_1 \cdot \text{Material} + a_2 \cdot \text{Bauernstruktur} + a_3 \cdot \text{Königsicherheit} \\ + a_4 \cdot \text{Springer_im_Zentrum} + a_5 \cdot \text{Läufer_Diagonalabdeckung} + \dots, \quad (6.3)$$

wobei das mit Abstand wichtigste Feature „Material“ nach der Formel

$$\text{Material} = \text{Material}(\text{eigenes Team}) - \text{Material}(\text{Gegner})$$

mit

$$\text{Material}(\text{Team}) = \text{Anz. Bauern}(\text{Team}) \cdot 100 + \text{Anz. Springer}(\text{Team}) \cdot 300 \\ + \text{Anz. Läufer}(\text{Team}) \cdot 300 + \text{Anz. Türme}(\text{Team}) \cdot 500 \\ + \text{Anz. Damen}(\text{Team}) \cdot 900$$

berechnet wird. Beim Material bewerten fast alle Schachprogramme ähnlich. Große Unterschiede gibt es jedoch bei allen weiteren Features, auf die wir hier nicht näher eingehen [Fra05, Lar00].

Im nächsten Schritt müssen nun die Gewichtungen a_i aller Features bestimmt werden. Diese werden nach Absprache mit dem Experten gefühlsmäßig festgelegt und dann nach jedem Spiel aufgrund positiver und negativer Erfahrungen verändert. Da dieser Prozess der Optimierung der Gewichte sehr aufwändig ist und außerdem die lineare Kombination der Features sehr eingeschränkt ist, bietet es sich an, für diese Aufgabe maschinelle Lernverfahren einzusetzen.

6.5.1 Lernen von Heuristiken

Wir wollen nun also die Gewichte a_i der Bewertungsfunktion $B(s)$ aus (6.3) automatisch optimieren. Bei diesem Ansatz wird der Experte nur noch nach den relevanten Features $f_1(s), \dots, f_n(s)$ für Spielzustände s befragt. Beim Schach sind dies Stellungen. Nun wird ein maschinelles Lernverfahren verwendet mit dem Ziel, eine möglichst optimale Bewertungsfunktion $B(f_1, \dots, f_n)$ zu finden. Man startet mit einer (durch das Lernverfahren bestimmten) initialen, zum Beispiel zufällig vorbelegten Bewertungsfunktion und lässt dann damit das Schachprogramm spielen. Am Ende jedes Spiels erfolgt durch das Ergebnis (Sieg, Niederlage, Remis) eine Bewertung. Aufgrund dieser Bewertung verändert nun das Lernverfahren die Bewertungsfunktion mit dem Ziel, beim nächsten Mal weniger Fehler zu machen. Im Prinzip wird hier das, was bei der manuellen Variante die Entwickler tun, um das System zu verbessern, automatisch von dem Lernverfahren erledigt.

So einfach sich dies anhört, so schwierig ist es in der Praxis. Ein zentrales Problem beim Verbessern der Stellungsbewertung aufgrund von gewonnenen oder verlorenen Partien ist heute bekannt unter dem Namen **Credit Assignment**. Man hat zwar am Ende des Spiels eine Bewertung des ganzen Spiels, aber keine Bewertung der einzelnen Züge. Der Agent macht also viele Aktionen und erhält erst am Ende ein positives oder negatives Feedback. Wie soll er nun den vielen Aktionen in der Vergangenheit dieses Feedback zuordnen? Und wie soll er dann seine Aktionen gegebenenfalls verbessern? Mit diesen Fragen beschäftigt sich das spannende junge Gebiet des **Lernens durch Verstärkung** (engl. reinforcement learning) (siehe Kap. 10).

Die weltbesten Schachcomputer arbeiten bis heute immer noch ohne Lernverfahren. Dafür gibt es zwei Gründe. Einerseits benötigen die bis heute entwickelten Verfahren zum Lernen durch Verstärkung bei großen Zustandsräumen noch sehr viel Rechenzeit. Andererseits sind aber die manuell erstellten Heuristiken der Hochleistungsschachcomputer schon sehr stark optimiert. Das heißt, dass nur ein sehr gutes Lernverfahren noch zu Verbesserungen führen kann. In den nächsten zehn Jahren wird vermutlich der Zeitpunkt kommen, an dem ein lernfähiger Schachcomputer Weltmeister wird.

6.6 Stand der Forschung

Um die Qualität der heuristischen Suchverfahren zu bewerten, möchte ich die Definition von Elaine Rich [Ric83] wiederholen:

Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

Kaum ein Test ist besser geeignet für die Entscheidung, ob ein Computerprogramm intelligent ist, als der direkte Vergleich von Computer und Mensch in einem Spiel wie Dame, Schach, Backgammon oder Go.

Schon 1950 wurden von Claude Shannon, Konrad Zuse und John von Neumann erste Schachprogramme vorgestellt, die allerdings nicht oder nur extrem langsam implementiert werden konnten. Nur wenige Jahre danach, 1955, schrieb Arthur Samuel ein Programm, das Dame spielen und mit einem einfachen Lernverfahren seine Parameter verbessern konnte. Er verwendete dazu den ersten speicherprogrammierbaren Computer von IBM, den IBM 701. Im Vergleich zu den heutigen Schachcomputern hatte er jedoch Zugriff auf eine große Zahl von archivierten Spielen, bei denen jeder einzelne Zug von Experten bewertet war. Damit verbesserte das Programm seine Bewertungsfunktion. Um eine noch weitere Verbesserung zu erreichen, ließ Samuel sein Programm gegen sich selbst spielen. Das Problem des Credit Assignment löste er auf einfache Weise. Für jede einzelne Stellung während eines Spiels vergleicht er die Bewertung durch die Funktion $B(s)$ mit der durch Alpha-Beta-Pruning berechneten Bewertung und verändert $B(s)$ entsprechend.

1961 besiegte sein Dame-Programm den viertbesten Damespieler der USA. Mit dieser bahnbrechenden Arbeit war Samuel seiner Zeit sicherlich um fast dreißig Jahre voraus.

Erst Anfang der Neunziger Jahre, als das Lernen durch Verstärkung aufkam, baute Gerald Tesauro mit ganz ähnlichen Verfahren ein lernfähiges Backgammon-Programm namens TD-Gammon, das auf Weltmeisterniveau spielte (siehe Kap. 10).

Heute existieren mehrere, teils kommerziell vertriebene Schachprogramme für PCs, die auf Großmeisterniveau spielen. Der Durchbruch gelang 1997, als IBM's Deep Blue mit 3,5 zu 2,5 gewonnenen Spielen den Schachweltmeister Gary Kasparov besiegte. Deep Blue konnte im Mittel etwa 12 Halbzüge mit AlphaBeta-Pruning und heuristischer Stellungsbewertung vorausberechnen.

Der derzeit leistungsfähigste Schachcomputer ist Hydra, ein Parallelrechner einer Firma in den Arabischen Emiraten. Die Software wurde von den Wissenschaftlern Christian Donninger (Österreich) und Ulf Lorenz (Deutschland) sowie dem deutschen Schach-Großmeister Christopher Lutz entwickelt. Hydra arbeitet mit 64 parallelen Xeon-Prozessoren mit je etwa 3 GHz Rechenleistung und 1 GByte Speicher. Für die Stellungsbewertung besitzt jeder Prozessor einen FPGA-Coprozessor (Field Programmable Gate Arrays). Dadurch wird es möglich, mit einer aufwändigen Bewertungsfunktion 200 Millionen Stellungen pro Sekunde zu bewerten.

Hydra kann mit dieser Technologie im Mittel etwa 18 Halbzüge vorausberechnen. In speziellen kritischen Situationen kann der Suchhorizont sogar bis auf 40 Halbzüge ausgedehnt werden. Offenbar liegt ein derartiger Horizont jenseits dessen, was die besten Großmeister schaffen, denn oftmals macht Hydra Züge, die Großmeister nicht nachvollziehen können, die aber letztlich zum Erfolg führen.

Hydra verwendet wenig spezielles Schach-Lehrbuchwissen, sondern Alpha-Beta-Suche mit relativ allgemeinen, bekannten Heuristiken und eine gute handkodierte Stellungsbewertung. Insbesondere ist Hydra nicht lernfähig. Verbesserungen werden von Spiel zu Spiel durch die Entwickler vorgenommen. Das Lernen wird dem Computer also immer noch abgenommen. Auch besitzt Hydra keine speziellen Planungsalgorithmen. Dass Hydra ohne Lernen arbeitet, ist ein Hinweis darauf, dass beim maschinellen Lernen trotz vieler Erfolge noch Forschungsbedarf besteht. Wie schon erwähnt werden wir in Kap. 10 und 8 hierauf näher eingehen.

Im Jahr 2009 gewann der Schachcomputer Pocket Fritz 4 auf einem PDA das Copa Mercosur Schachturnier in Buenos Aires mit neun Siegen und einem Unentschieden gegen zehn excellente menschliche Schachspieler, darunter drei Großmeister. Obwohl nur wenig über die Softwarestruktur bekannt ist, zeigt sie ganz klar einen Trend weg von roher Rechenleistung und hin zu mehr Intelligenz auf. Pocket Fritz 4 spielt auf Großmeister-Niveau und ist vergleichbar, wenn nicht sogar besser als Hydra. Laut dem Entwickler Stanislav Tsukrov ([Wik13] → HIARCS), kann Pocket Fritz mit seinem Schach-Suchalgorithmus HIARCS 13 weniger als 20.000 Stellungen pro Sekunde evaluieren. Dies ist um etwa den Faktor 10.000 langsamer als Hydra. Daraus lässt sich schließen, dass HIARCS 13 wesentlich bessere Heuristiken zur Reduktion des effektiven Verzweigungsfaktors benutzt als Hydra

und daher mit gutem Recht als intelligenter bezeichnet werden kann. HIARCS ist übrigens die Abkürzung für *Higher Intelligence Auto Response Chess System*.

Auch wenn heute praktisch alle Menschen chancenlos sind gegen die besten Schachcomputer, so gibt es noch viele Herausforderungen für die KI. Zum Beispiel Go. Bei diesem alten japanischen Spiel auf einem quadratischen Brett mit 361 Feldern, 181 weißen und 180 schwarzen Steinen liegt der effektive Verzweigungsfaktor bei etwa 300. Schon nach 4 Halbzügen ergeben sich etwa $8 \cdot 10^9$ Stellungen. Alle bekannten klassischen Spielbaum-Suchverfahren sind bei dieser Komplexität chancenlos gegen gute menschliche Go-Spieler. Die Experten sind sich einig, dass hier „wirklich intelligente“ Verfahren gefragt sind. Das kombinatorische Aufzählen aller Möglichkeiten ist die falsche Methode. Vielmehr werden hier Verfahren benötigt, die Muster auf dem Brett erkennen, langsame Entwicklungen verfolgen und schnelle „intuitive“ Entscheidungen treffen können. Ähnlich wie beim Erkennen von Objekten auf komplexen Bildern sind wir Menschen den heutigen Computerprogrammen noch meilenweit überlegen. Wir verarbeiten das Bild als ganzes hochparallel, wogegen der Computer nacheinander die Millionen von Pixeln verarbeitet und große Schwierigkeiten hat, in der Fülle der Pixel das Wesentliche zu erkennen. Das Programm „The many faces of Go“ kann 1100 verschiedene Muster erkennen und kennt 200 verschiedene Spielstrategien. Alle Go-Programme haben aber noch große Probleme, zu erkennen, ob eine Gruppe von Steinen tot oder lebendig ist, beziehungsweise wo sie dazwischen einzuordnen ist.

6.7 Übungen

Aufgabe 6.1

- Beweisen Sie Satz 6.1, das heißt, dass bei einem Baum mit großem konstantem Verzweigungsfaktor b auf der letzten Ebene in Tiefe d fast alle Knoten liegen.
- Zeigen Sie, dass dies nicht immer gilt, wenn der effektive Verzweigungsfaktor groß aber variabel ist.

Aufgabe 6.2

- Berechnen Sie den mittleren Verzweigungsfaktor beim 8-Puzzle bei uninformerter Suche ohne Zyklencheck. Der mittlere Verzweigungsfaktor ist der Verzweigungsfaktor, den ein Baum mit gleicher Knotenzahl auf der letzten Ebene, konstantem Verzweigungsfaktor und gleicher Tiefe hätte.
- Berechnen Sie den mittleren Verzweigungsfaktor beim 8-Puzzle bei uninformerter Suche bei Vermeidung von Zyklen der Länge 2.

Aufgabe 6.3

- Worin liegt der Unterschied zwischen dem mittleren und dem effektiven Verzweigungsfaktor (Definition 6.2)?

- b) Warum ist der effektive Verzweigungsfaktor zur Analyse und zum Vergleich der Rechenzeiten von Suchalgorithmen meist besser geeignet als der mittlere?
- c) Zeigen Sie, dass für einen stark verzweigenden Baum mit n Knoten und Tiefe d der effektive Verzweigungsfaktor \bar{b} etwa gleich dem mittleren Verzweigungsfaktor und damit gleich $\sqrt[d]{n}$ ist.

Aufgabe 6.4

- a) Berechnen Sie die Größe des Zustandsraumes 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, bestehend aus den Zuständen (Knoten) und den Aktionen (Kanten), beim 3-Puzzle in zwei zusammenhängende Teilgraphen zerfällt, zwischen denen keine Verbindung besteht.

Aufgabe 6.5 Suchen Sie (manuell) mit Breitensuche für das 8-Puzzle einen Pfad vom Startknoten  zum Zielknoten .

Aufgabe 6.6 \Rightarrow

- a) Programmieren Sie in einer Programmiersprache Ihrer Wahl Breitensuche, Tiefensuche und Iterative Deepening und testen diese am Beispiel des 8-Puzzle.
- b) Warum macht die Verwendung der Tiefensuche beim 8-Puzzle wenig Sinn?

Aufgabe 6.7

- a) Zeigen Sie, dass die Breitensuche bei konstanten Kosten für alle Aktionen garantiert die kürzeste Lösung findet.
- b) Zeigen Sie, dass dies bei variierenden Kosten nicht gilt.

Aufgabe 6.8 Suchen Sie (manuell) mit A^* -Suche für das 8-Puzzle einen Pfad vom Startknoten  zum Zielknoten .

- a) unter Verwendung der Heuristik h_1 (Abschn. 6.3.4),
 b) unter Verwendung der Heuristik h_2 (Abschn. 6.3.4).

Aufgabe 6.9 Konstruieren Sie für den Städtegraphen aus Abb. 6.14 den Suchbaum mit A^* -Suche und verwenden Sie als Heuristik die Luftlinienentfernung nach Ulm. Starten Sie in Bern mit Ziel Ulm. Beachten Sie, dass jeder Ort höchstens einmal pro Pfad auftritt.

Aufgabe 6.10 \Rightarrow Programmieren Sie in einer Programmiersprache Ihrer Wahl die A^* -Suche mit den Heuristiken h_1 und h_2 und testen diese am Beispiel des 8-Puzzle.

Aufgabe 6.11 * Geben Sie für die Funktion HEURISTISCHESUCHE je eine heuristische Bewertungsfunktion für Zustände an, mit der sich Tiefensuche und Breitensuche realisieren lassen.

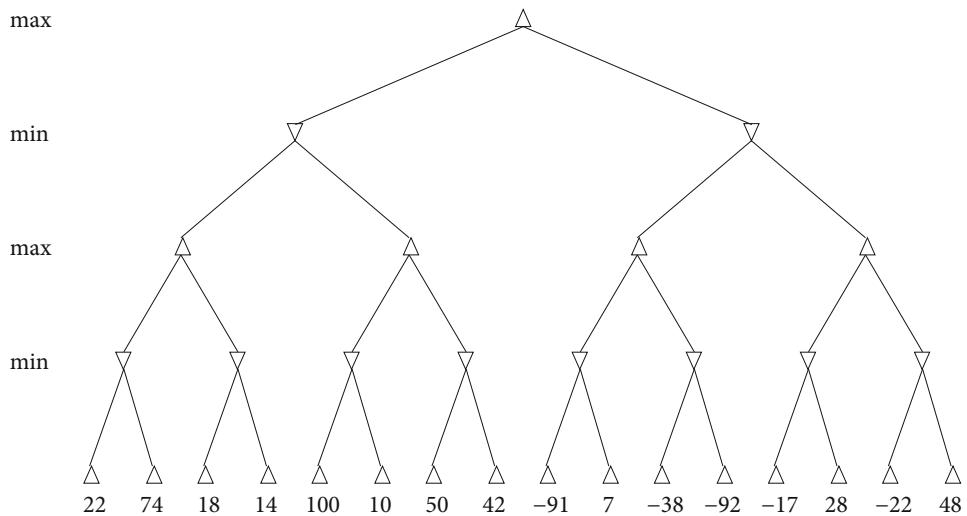


Abb. 6.21 Minimax-Suchbaum

Aufgabe 6.12 Welchen Bezug hat das Bild mit dem Ehepaar an der Schlucht aus Abb. 6.13 zu einer zulässigen Heuristik?

Aufgabe 6.13 Zeigen Sie, dass die Heuristiken h_1 und h_2 für das 8-Puzzle aus Abschn. 6.3.4 zulässig sind.

Aufgabe 6.14

- Gegeben ist der Suchbaum eines 2-Spieler-Spiels in Abb. 6.21 mit den Bewertungen aller Blattknoten. Verwenden Sie Minimax-Suche mit α - β -Pruning von links nach rechts. Streichen Sie alle nicht besuchten Knoten und geben Sie für jeden inneren Knoten die optimale resultierende Bewertung an. Markieren Sie den gewählten Pfad.
- Testen Sie sich mit Hilfe des Applets von P. Winston [[Win](#)].

Dass eine zweiwertige Logik beim Schließen im Alltag zu Problemen führt, haben wir in Kap. 4 an Hand des Tweety-Problems aufgezeigt. In diesem Beispiel führen die Aussagen *Tweety ist ein Pinguin*, *Alle Vögel können fliegen* und *Pinguine sind Vögel* zu der Folgerung *Tweety kann fliegen*. Interessant wäre zum Beispiel eine Sprache, in der es möglich ist, die Aussage *Fast alle Vögel können fliegen* zu formalisieren und darauf dann Inferenzen durchzuführen. Die Wahrscheinlichkeitsrechnung stellt hierfür eine bewährte Methode bereit, denn durch die Angabe eines Wahrscheinlichkeitswertes lässt sich die Unsicherheit über das Fliegen von Vögeln gut modellieren. Wir werden zeigen, dass etwa eine Aussage wie *99 % aller Vögel können fliegen* zusammen mit Wahrscheinlichkeitslogik zu korrekten Schlüssen führt.

Das Schließen mit Unsicherheit bei beschränkten Ressourcen spielt im Alltag und auch in vielen technischen Anwendungen der KI eine große Rolle. Ganz wichtig sind hierbei heuristische Verfahren, wie wir sie in Kap. 6 schon besprochen haben. Zum Beispiel auf der Suche nach einem Parkplatz im Stadtverkehr verwenden wir heuristische Techniken. Heuristiken allein genügen aber oft nicht, insbesondere wenn eine schnelle Entscheidung bei unvollständigem Wissen gefordert ist, wie folgendes Beispiel aufzeigt. Ein Fußgänger überquert eine Straße und ein Auto nähert sich schnell. Um einen folgenschweren Unfall zu verhindern, muss der Fußgänger nun sehr schnell handeln. Er ist in dieser Situation nicht in der Lage, sich vollständige Informationen über den Zustand der Welt zu beschaffen, die er für die in Kap. 6 behandelten Verfahren benötigen würde. Er muss also schnell zu einer unter den gegebenen Randbedingungen (wenig Zeit und wenig, eventuell unsicheres, Wissen) optimalen Entscheidung kommen. Denn wenn er zu lange nachdenkt, wird es gefährlich. Hier und auch in vielen ähnlichen Situationen (siehe Abb. 7.1) wird eine Methode zum Schließen mit unsicherem und unvollständigem Wissen benötigt.

An einem einfachen Beispiel aus der medizinischen Diagnose wollen wir verschiedene Möglichkeiten des Schließens mit Unsicherheit untersuchen. Hat ein Patient Schmerzen im rechten Unterbauch und erhöhten Leukozytenwert, so besteht der Verdacht auf eine akute Blinddarmentzündung (Appendizitis). Wir modellieren diesen Zusammenhang mit

Abb. 7.1 „Lass uns mal in Ruhe überlegen, was wir nun machen!“



Aussagenlogik durch die Formel

$$\text{Bauchschmerzen re. u.} \wedge \text{Leukozyten} > 10.000 \rightarrow \text{Blinddarmentzündung}$$

Wenn wir nun noch wissen, dass

$$\text{Bauchschmerzen re. u.} \wedge \text{Leukozyten} > 10.000$$

gilt, so können wir mit Modus Ponens *Blinddarmentzündung* ableiten. Diese Modellierung ist offenbar zu grob. Dies haben 1976 Shortliffe und Buchanan beim Bau Ihres medizinischen Expertensystems MYCIN erkannt [Sho76]. Sie führten mit Hilfe der so genannten Certainty Factors einen Kalkül ein, der es ihnen erlaubte, den Grad der Sicherheit von Fakten und Regeln zu repräsentieren. Einer Regel $A \rightarrow B$ wird ein Sicherheitsfaktor β zugeordnet. Die Semantik einer Regel $A \rightarrow_{\beta} B$ wurde definiert über die bedingte Wahrscheinlichkeit $P(B | A) = \beta$. Im obigen Beispiel könnte die Regel dann

$$\text{Bauchschm. re. u.} \wedge \text{Leukozyten} > 10.000 \rightarrow_{0,6} \text{Blinddarmentzündung}$$

lauten. Zum Schließen mit derartigen Regeln wurden Formeln für die Verknüpfung der Faktoren von Regeln angegeben. Es stellte sich jedoch heraus, dass der Kalkül bezüglich dieser Verknüpfungsregeln inkorrekt angelegt war, denn es konnten mit ihm inkonsistente Ergebnisse abgeleitet werden.

Wie schon in Kap. 4 erwähnt, wurde auch versucht, mit Hilfe von nichtmonotonen Logiken und Defaultlogik die genannten Probleme zu lösen, was aber letztlich nicht zum Erfolg führte. Die Dempster-Schäfer-Theorie ordnet einer logischen Aussage A eine Glau-bensfunktion (engl. belief function) $Bel(A)$ zu, deren Wert den Grad der Evidenz für die Wahrheit von A angibt. Aber auch dieser Formalismus hat Schwächen, wie in [Pea88] auf S. 447 an Hand einer Variante des Tweety-Beispiels gezeigt wird. Auch die vor allem in der Regelungstechnik erfolgreiche Fuzzy-Logik zeigt beim Schließen mit Unsicherheit in komplexeren Anwendungen erhebliche Schwächen [Elk93] auf.

Seit etwa Mitte der Achtziger-Jahre findet die Wahrscheinlichkeitsrechnung immer mehr Einzug in die KI [Pea88, Che85, Whi96, Jen01]. In dem Gebiet des Schließens mit Bayes-Netzen beziehungsweise subjektiven Wahrscheinlichkeiten hat sie sich mittlerweile einen festen Platz unter den erfolgreichen KI-Techniken gesichert. Statt der aus der Logik bekannten Implikation (materiale Implikation) werden hier bedingte Wahrscheinlichkeiten verwendet, welche das im Alltag verwendete kausale Schließen wesentlich besser modellieren. Das Schließen mit Wahrscheinlichkeiten profitiert stark davon, dass die Wahrscheinlichkeitstheorie ein hunderte Jahre altes sehr gut fundiertes Teilgebiet der Mathematik ist.

Wir werden in diesem Kapitel einen eleganten, aber für ein Lehrbuch etwas ungewöhnlichen Zugang zu diesem Gebiet wählen. Nach einer kurzen Einführung in die wichtigsten hier benötigten Grundlagen zum Rechnen mit Wahrscheinlichkeiten starten wir mit einem einfachen, aber wichtigen Beispiel, bei dem Schließen mit unsicherem und unvollständigem Wissen gefragt ist. In ganz natürlicher, fast zwingender, Weise werden wir hierbei zur Methode der maximalen Entropie (MaxEnt) geführt. Dann zeigen wir an Hand des medizinischen Expertensystems LEXMED die Tauglichkeit dieser Methode für die Praxis auf. Schließlich führen wir das heute weit verbreitete Schließen mit Bayes-Netzen ein und zeigen den Zusammenhang zwischen den beiden Methoden auf.

7.1 Rechnen mit Wahrscheinlichkeiten

Der mit Wahrscheinlichkeitsrechnung vertraute Leser kann auf diesen Abschnitt verzichten. Für alle Anderen geben wir hier einen Schnelleinstieg und verweisen auf einschlägige Lehrbücher wie zum Beispiel [GT96, Hüb03].

Wahrscheinlichkeiten eignen sich für die Modellierung des Schließens mit Unsicherheit besonders gut. Ein Grund hierfür ist ihre intuitiv einfache Interpretierbarkeit, was an folgendem elementaren Beispiel gut erkennbar ist.

Beispiel 7.1

Beim einmaligen Würfeln mit einem Spielwürfel (Versuch) ist die Wahrscheinlichkeit für das Ereignis „Würfeln einer Sechs“ gleich $1/6$, wogegen die Wahrscheinlichkeit für das Ereignis „Würfeln einer ungeraden Zahl“ gleich $1/2$ ist.

Definition 7.1

Sei Ω die zu einem Versuch gehörende endliche Menge von **Ereignissen**. Jedes Ereignis $\omega \in \Omega$ steht für einen möglichen Ausgang des Versuchs. Schließen sich die Ereignisse $\omega_i \in \Omega$ gegenseitig aus, decken aber alle möglichen Ausgänge des Versuchs ab, so werden diese **Elementarereignisse** genannt.

Beispiel 7.2

Beim einmaligen Würfeln mit einem Spielwürfel ist

$$\Omega = \{1, 2, 3, 4, 5, 6\},$$

denn keine zwei dieser Ereignisse können gleichzeitig auftreten. Das Würfeln einer geraden Zahl ($\{2, 4, 6\}$) ist dann kein Elementarereignis, genauso wie das Würfeln einer Zahl kleiner als 5 ($\{1, 2, 3, 4\}$), denn $\{2, 4, 6\} \cap \{1, 2, 3, 4\} = \{2, 4\} \neq \emptyset$.

Mit zwei Ereignissen A und B ist $A \cup B$ auch ein Ereignis. Ω selbst wird als **sicheres Ereignis** und die leere Menge \emptyset als **unmögliches Ereignis** bezeichnet.

Im Folgenden werden wir die aussagenlogische Schreibweise für Mengenoperationen verwenden. Das heißt für die Menge $A \cap B$ schreiben wir $A \wedge B$. Dies ist nicht nur eine syntaktische Transformation, sondern es ist auch semantisch korrekt, denn der Durchschnitt von zwei Mengen ist definiert wie folgt

$$x \in A \cap B \Leftrightarrow x \in A \wedge x \in B.$$

Da dies die Semantik von $A \wedge B$ ist, können und werden wir diese Schreibweise verwenden. Auch für die anderen Mengenoperationen Vereinigung und Komplement gilt dies und wir werden, wie in folgender Tabelle dargestellt, die aussagenlogische Schreibweise verwenden.

Mengenschreibweise	Aussagenlogik	Beschreibung
$A \cap B$	$A \wedge B$	Schnittmenge/und
$A \cup B$	$A \vee B$	Vereinigung/oder
\bar{A}	$\neg A$	Komplement/Negation
Ω	w	Sicheres Ereignis/wahr
\emptyset	f	Unmögliches Ereignis/falsch

Die hier verwendeten Variablen (z. B. A , B , etc.) heißen in der Wahrscheinlichkeitsrechnung **Zufallsvariablen**. Wir werden hier nur diskrete Zufallsvariablen mit endlichem Wertebereich verwenden. Die Variable *Augenzahl* beim Würfeln ist diskret mit den Werten

1, 2, 3, 4, 5, 6. Die Wahrscheinlichkeit, eine Fünf oder eine Sechs zu würfeln ist gleich 1/3. Dies lässt sich beschreiben durch

$$P(\text{Augenzahl} \in \{5, 6\}) = P(\text{Augenzahl} = 5 \vee \text{Augenzahl} = 6) = 1/3.$$

Der Begriff der Wahrscheinlichkeit soll eine möglichst objektive Beschreibung unseres „Glaubens“ oder unserer „Überzeugung“ über den Ausgang eines Versuchs liefern. Als numerische Werte sollen alle reellen Zahlen im Intervall [0, 1] möglich sein, wobei 0 die Wahrscheinlichkeit für das unmögliche Ereignis und 1 die Wahrscheinlichkeit für das sichere Ereignis sein soll. Dies wird erreicht durch die folgende Definition.

Definition 7.2

Sei $\Omega = \{\omega_1, \omega_2, \dots, \omega_n\}$ endlich. Es sei kein Elementarereignis bevorzugt, d. h. man setzt eine Symmetrie bezüglich der Häufigkeit des Auftretens aller Elementarereignisse voraus. Die **Wahrscheinlichkeit** $P(A)$ des Ereignisses A ist dann

$$P(A) = \frac{|A|}{|\Omega|} = \frac{\text{Anzahl der für } A \text{ günstigen Fälle}}{\text{Anzahl der möglichen Fälle}}.$$

Es folgt sofort, dass jedes Elementarereignis die Wahrscheinlichkeit $1/|\Omega|$ hat. Die Voraussetzung der Gleichwahrscheinlichkeit der Elementarereignisse nennt man **Laplace-Annahme** und die damit berechneten Wahrscheinlichkeiten **Laplace-Wahrscheinlichkeiten**. Diese Definition stößt an ihre Grenzen, wenn die Zahl der Elementarereignisse unendlich wird. Da wir hier aber nur endliche Ereignisräume betrachten werden, stellt dies kein Problem dar. Zur Beschreibung von Ereignissen verwenden wir Variablen mit entsprechend vielen Werten. Zum Beispiel kann eine Variable *Augenfarbe* die Werte *grün*, *blau*, *braun* annehmen. *Augenfarbe = blau* beschreibt dann ein Ereignis, denn es handelt sich um eine Aussage mit den Wahrheitswerten *w* oder *f*. Bei binären (booleschen) Variablen ist schon die Variable selbst eine Aussage. Es genügt hier also zum Beispiel die Angabe von $P(\text{JohnRuftAn})$ statt $P(\text{JohnRuftAn} = w)$.

Beispiel 7.3

Die Wahrscheinlichkeit, eine gerade Augenzahl zu würfeln ist nach dieser Definition

$$P(\text{Augenzahl} \in \{2, 4, 6\}) = \frac{|\{2, 4, 6\}|}{|\{1, 2, 3, 4, 5, 6\}|} = \frac{3}{6} = \frac{1}{2}.$$

Direkt aus der Definition folgen einige wichtige Regeln:

Satz 7.1

1. $P(\Omega) = 1$.
2. $P(\emptyset) = 0$, d. h. das unmögliche Ereignis hat die Wahrscheinlichkeit 0.
3. Für paarweise unvereinbare Ereignisse A und B gilt $P(A \vee B) = P(A) + P(B)$.
4. Für zwei zueinander komplementäre Ereignisse A und $\neg A$ gilt $P(A) + P(\neg A) = 1$.
5. Für beliebige Ereignisse A und B gilt $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.
6. Für $A \subseteq B$ gilt $P(A) \leq P(B)$.
7. Sind A_1, \dots, A_n die Elementarereignisse, so gilt $\sum_{i=1}^n P(A_i) = 1$ (Normierungsbedingung).

Der Ausdruck $P(A \wedge B)$ oder auch $P(A, B)$ steht für die Wahrscheinlichkeit des Ereignisses $A \wedge B$. Oft interessieren wir uns für die Wahrscheinlichkeiten aller Elementarereignisse, das heißt aller Kombinationen aller Werte der Variablen A und B . Bei den zweierwerten Variablen A und B sind dies $P(A, B)$, $P(A, \neg B)$, $P(\neg A, B)$, $P(\neg A, \neg B)$. Den Vektor

$$(P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

bestehend aus diesen vier Werten nennt man **Verteilung** oder **Wahrscheinlichkeitsverteilung** (engl. joint probability distribution) der Variablen A und B . Er wird abgekürzt mit $P(A, B)$. Schön anschaulich lässt sich die Verteilung im Fall von zwei Variablen in Form einer Tabelle (Matrix) wie folgt darstellen:

$P(A, B)$	$B = w$	$B = f$
$A = w$	$P(A, B)$	$P(A, \neg B)$
$A = f$	$P(\neg A, B)$	$P(\neg A, \neg B)$

Bei den d Variablen X_1, \dots, X_d mit je n Werten enthält die Verteilung die Werte $P(X_1 = x_1, \dots, X_d = x_d)$ und x_1, \dots, x_d nehmen jeweils n verschiedene Werte an. Die Verteilung lässt sich daher als d -dimensionale Matrix mit insgesamt n^d Elementen darstellen. Aufgrund der Normierungsbedingung aus Satz 7.1 ist jedoch einer dieser n^d Werte redundant und die Verteilung wird durch $n^d - 1$ Werte eindeutig charakterisiert.

7.1.1 Bedingte Wahrscheinlichkeiten

Beispiel 7.4

In der Doggenriedstraße in Weingarten wird die Geschwindigkeit von 100 Fahrzeugen gemessen. Bei jeder Messung wird protokolliert, ob der Fahrer Student ist oder nicht. Die Ergebnisse sind:

Ereignis	Häufigkeit	Relative Häufigkeit
Fahrzeug beobachtet	100	1
Fahrer ist Student (S)	30	0,3
Geschwindigkeit zu hoch (G)	10	0,1
Fahrer ist Student und Geschw. zu hoch ($S \wedge G$)	5	0,05

Wir stellen die Frage: *Fahren Studenten häufiger zu schnell als der Durchschnitt bzw. als Nichtstudenten?*¹

Die Antwort wird gegeben durch die Wahrscheinlichkeit

$$P(G|S) = \frac{|\text{Fahrer ist Student und Geschw. zu hoch}|}{|\text{Fahrer ist Student}|} = \frac{5}{30} = \frac{1}{6} \approx 0,17$$

für zu schnelles Fahren unter der Bedingung, dass der Fahrer Student ist. Diese unterscheidet sich offensichtlich von der a-priori-Wahrscheinlichkeit $P(G) = 0,1$ für zu schnelles Fahren. Bei der a-priori-Wahrscheinlichkeit wird der Ereignisraum nicht durch Zusatzbedingungen eingeschränkt.

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

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

An Beispiel 7.4 erkennt man, dass im Fall eines endlichen Ereignisraumes die bedingte Wahrscheinlichkeit $P(A|B)$ aufgefasst werden kann als die Wahrscheinlichkeit von A und B , wenn man nur das Ereignis B betrachtet, d. h. als

$$P(A|B) = \frac{|A \wedge B|}{|B|}.$$

¹ Die berechneten Wahrscheinlichkeiten können nur dann für weitergehende Aussagen benutzt werden, wenn die gemessene Stichprobe (100 Fahrzeuge) repräsentativ ist. Andernfalls können nur Aussagen über die beobachteten 100 Fahrzeuge gemacht werden.

Diese Formel lässt sich einfach herleiten unter Verwendung von Definition 7.2

$$P(A | B) = \frac{P(A \wedge B)}{P(B)} = \frac{\frac{|A \wedge B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{|A \wedge B|}{|B|}.$$

Definition 7.4

Gilt für zwei Ereignisse A und B

$$P(A | B) = P(A),$$

so nennt man diese Ereignisse unabhängig.

A und B sind also unabhängig, wenn die Wahrscheinlichkeit für das Ereignis A nicht durch das Ereignis B beeinflusst wird.

Satz 7.2

Für unabhängige Ereignisse A und B folgt aus der Definition

$$P(A \wedge B) = P(A) \cdot P(B).$$

Beispiel 7.5

Beim Würfeln mit zwei Würfeln ist die Wahrscheinlichkeit für zwei Sechsen $1/36$, wenn die beiden Würfel unabhängig sind, denn

$$P(W_1 = 6 \wedge W_2 = 6) = P(W_1 = 6) \cdot P(W_2 = 6) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36},$$

wobei die erste Gleichung nur gilt, wenn die beiden Würfel unabhängig sind. Fällt z. B. durch magische Kräfte Würfel 2 immer gleich wie Würfel 1, so gilt

$$P(W_1 = 6 \wedge W_2 = 6) = \frac{1}{6}.$$

Kettenregel

Auflösen der Definition der bedingten Wahrscheinlichkeit nach $P(A \wedge B)$ ergibt die so genannte Produktregel

$$P(A \wedge B) = P(A | B)P(B),$$

die wir sofort auf den Fall von n Variablen verallgemeinern. Durch wiederholte Anwendung obiger Regel erhalten wir die **Kettenregel**

$$\begin{aligned} \mathbf{P}(X_1, \dots, X_n) &= \mathbf{P}(X_n | X_1, \dots, X_{n-1}) \cdot \mathbf{P}(X_1, \dots, X_{n-1}) \\ &= \mathbf{P}(X_n | X_1, \dots, X_{n-1}) \cdot \mathbf{P}(X_{n-1} | X_1, \dots, X_{n-2}) \cdot \mathbf{P}(X_1, \dots, X_{n-2}) \\ &= \mathbf{P}(X_n | X_1, \dots, X_{n-1}) \cdot \mathbf{P}(X_{n-1} | X_1, \dots, X_{n-2}) \cdots \mathbf{P}(X_2 | X_1) \cdot \mathbf{P}(X_1) \\ &= \prod_{i=1}^n \mathbf{P}(X_i | X_1, \dots, X_{i-1}), \end{aligned} \quad (7.1)$$

mit der man die Verteilung als ein Produkt bedingter Wahrscheinlichkeiten darstellen kann. Da die Kettenregel für alle Werte der Variablen X_1, \dots, X_n gilt, wurde sie mit Hilfe des Symbols \mathbf{P} für die Verteilung formuliert.

Marginalisierung

Wegen $A \Leftrightarrow (A \wedge B) \vee (A \wedge \neg B)$ gilt für zweiwertige Variablen A und B

$$P(A) = P((A \wedge B) \vee (A \wedge \neg B)) = P(A \wedge B) + P(A \wedge \neg B).$$

Durch Summation über die beiden Werte von B wird die Variable B eliminiert. Analog lässt sich für beliebige Variablen X_1, \dots, X_d eine Variable, zum Beispiel X_d , durch Summation über alle ihre Werte eliminieren. Es gilt

$$P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}) = \sum_{x_d} P(X_1 = x_1, \dots, X_{d-1} = x_{d-1}, X_d = x_d).$$

Die Anwendung dieser Formel wird Marginalisierung genannt. Diese Summation kann mit den Variablen X_1, \dots, X_{d-1} so lange fortgesetzt werden, bis nur noch eine Variable übrig bleibt. Auch ist die Marginalisierung auf die Verteilung $\mathbf{P}(X_1, \dots, X_d)$ anwendbar. Die resultierende Verteilung $\mathbf{P}(X_1, \dots, X_{d-1})$ wird **Randverteilung** genannt, denn die Marginalisierung ist vergleichbar mit der Projektion eines Quaders auf eine Seitenfläche. Hier wird das dreidimensionale Objekt auf einen „Rand“ des Quaders, das heißt auf eine zweidimensionale Menge, abgebildet. In beiden Fällen wird die Dimension um eins reduziert.

Beispiel 7.6

Wir betrachten die Menge aller Patienten, die mit akuten Bauchschmerzen zum Arzt kommen. Bei diesen wird der Leukozytenwert gemessen, welcher ein Maß für die relative Häufigkeit der weißen Blutkörperchen im Blut ist. Wir definieren die Variable *Leuko*, welche genau dann wahr ist, wenn der Leukozytenwert größer als 10.000 ist. Dies deutet auf eine Entzündung im Körper hin. Außerdem definieren wir die Variable *App*, welche angibt, ob der Patient eine Appendizitis, das heißt einen entzündeten Blinddarm, hat. In folgender Tabelle ist die Verteilung $\mathbf{P}(App, Leuko)$ dieser beiden Variablen angegeben:

$P(App, Leuko)$	App	$\neg App$	Gesamt
Leuko	0,23	0,31	0,54
$\neg Leuko$	0,05	0,41	0,46
Gesamt	0,28	0,72	1

In der letzten Spalte und der letzten Zeile sind jeweils die Summen über eine Spalte beziehungsweise Zeile angegeben. Diese Summen sind durch Marginalisierung entstanden. Zum Beispiel liest man ab

$$P(Leuko) = P(App, Leuko) + P(\neg App, Leuko) = 0,54.$$

Die angegebene Verteilung $P(App, Leuko)$ könnte zum Beispiel aus einer Erhebung an deutschen Arztpraxen stammen. Daraus können wir nun die bedingte Wahrscheinlichkeit

$$P(Leuko | App) = \frac{P(Leuko, App)}{P(App)} = 0,82$$

berechnen, welche uns sagt, dass etwa 82 % aller Appendizitisfälle zu hohem Leukozytenwert führen. Derartige Werte werden in der medizinischen Literatur publiziert. Nicht publiziert wird hingegen die bedingte Wahrscheinlichkeit $P(App | Leuko)$, welche für die Diagnose der Appendizitis eigentlich viel hilfreicher wäre. Um dies zu verstehen, leiten wir zuerst eine einfache, aber sehr wichtige Formel her.

Die Bayes-Formel

Vertauschen von A und B in Definition 7.3 führt zu

$$P(A | B) = \frac{P(A \wedge B)}{P(B)} \quad \text{und} \quad P(B | A) = \frac{P(A \wedge B)}{P(A)}.$$

Durch Auflösen der beiden Gleichungen nach $P(A \wedge B)$ und Gleichsetzen erhält man die **Bayes-Formel**

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}, \tag{7.2}$$

welche wir sofort auf das Appendizitis-Beispiel anwenden und

$$P(App | Leuko) = \frac{P(Leuko | App) \cdot P(App)}{P(Leuko)} = \frac{0,82 \cdot 0,28}{0,54} = 0,43 \tag{7.3}$$

erhalten. Warum wird nun $P(Leuko | App)$ publiziert, $P(App | Leuko)$ aber nicht?

Unter der Annahme, dass sich eine Appendizitis, unabhängig von der Rasse, im Organismus aller Menschen ähnlich auswirkt, ist $P(Leuko | App)$ ein universeller, weltweit gültiger Wert. An (7.3) erkennt man, dass $P(App | Leuko)$ nicht universell ist, denn dieser Wert wird beeinflusst durch die A-priori-Wahrscheinlichkeiten $P(App)$ und $P(Leuko)$.

Beide können je nach den Lebensumständen variieren. Zum Beispiel hängt $P(\text{Leuko})$ davon ab, ob es in einer Personengruppe viele oder eher wenige Entzündungskrankheiten gibt. In den Tropen könnte sich der Wert eventuell deutlich von dem in kalten Regionen unterscheiden. Die Bayes-Formel macht es uns aber einfach, aus dem universell gültigen Wert $P(\text{Leuko} | \text{App})$ den für die Diagnose relevanten Wert $P(\text{App} | \text{Leuko})$ zu berechnen.

Bevor wir dieses Beispiel weiter vertiefen und in Abschn. 7.3 zu einem medizinischen Expertensystem für Appendizitis ausbauen, müssen wir zuerst den dafür benötigten probabilistischen Inferenzmechanismus einführen.

7.2 Die Methode der Maximalen Entropie

Am Beispiel einer einfachen Schlussfolgerung werden wir nun aufzeigen, wie man mit Hilfe der Wahrscheinlichkeitsrechnung einen Kalkül zum Schließen bei unsicherem Wissen realisieren kann. Wir werden allerdings bald sehen, dass die ausgetretenen probabilistischen Pfade schnell zu Ende sind. Wenn nämlich zu wenig Wissen für die Lösung der notwendigen Gleichungen vorhanden ist, sind neue Ideen gefragt. Der amerikanische Physiker E.T. Jaynes leistete hierzu in den fünfziger Jahren Pionierarbeit. Er forderte, dass bei fehlendem Wissen die Entropie der gesuchten Wahrscheinlichkeitsverteilung zu maximieren ist und wendete dieses Prinzip in [Jay57, Jay03] auf viele Beispiele an. Diese Methode wurde dann später weiterentwickelt [Che83, Nil86, Kan89, KK92] und ist heute ausgereift und technisch anwendbar, was wir in Abschn. 7.3 am Beispiel des LEXMED-Projektes zeigen werden.

7.2.1 Eine Inferenzregel für Wahrscheinlichkeiten

Wir wollen nun eine Inferenzregel für unsicheres Wissen analog zum klassischen Modus Ponens herleiten. Aus dem Wissen einer Aussage A und einer Regel $A \Rightarrow B$ soll auf B geschlossen werden. Kurz formuliert heißt das

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

Die Verallgemeinerung auf Wahrscheinlichkeitsregeln ergibt

$$\frac{P(A) = \alpha, P(B | A) = \beta}{P(B) = ?}.$$

Gegeben sind also die zwei Wahrscheinlichkeitswerte α, β und gesucht ist ein Wert für $P(B)$. Durch Marginalisierung erhalten wir aus der Verteilung die gesuchte 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, aber der Wert $P(B|\neg A)$ ist unbekannt. Mit klassischer Wahrscheinlichkeitsrechnung lässt sich hier keine genaue Aussage über $P(B)$ machen, allenfalls kann man abschätzen $P(B) \geq P(B|A) \cdot P(A)$.

Wir betrachten nun die Verteilung

$$\mathbf{P}(A, B) = (P(A, B), P(A, \neg B), P(\neg A, B), P(\neg A, \neg B))$$

und führen zur Abkürzung die 4 Unbekannten

$$\begin{aligned} p_1 &= P(A, B) \\ p_2 &= P(A, \neg B) \\ p_3 &= P(\neg A, B) \\ p_4 &= P(\neg A, \neg B) \end{aligned}$$

ein. Diese vier Parameter bestimmen die Verteilung. Sind sie alle bekannt, so lässt sich jede Wahrscheinlichkeit für die beiden Variablen A und B berechnen. Zur Berechnung dieser Werte werden vier Gleichungen benötigt. Eine Gleichung ist in Form der Normierungsbedingung

$$p_1 + p_2 + p_3 + p_4 = 1$$

schon bekannt. Es werden daher noch drei Gleichungen benötigt. In unserem Beispiel sind aber nur zwei Gleichungen bekannt.

Aus den gegebenen Werten $P(A) = \alpha$ und $P(B|A) = \beta$ berechnen wir

$$P(A, B) = P(B|A) \cdot P(A) = \alpha\beta$$

und

$$P(A) = P(A, B) + P(A, \neg B).$$

Daraus lässt sich folgendes Gleichungssystem aufstellen und soweit möglich auflösen:

$$p_1 = \alpha\beta \tag{7.4}$$

$$p_1 + p_2 = \alpha \tag{7.5}$$

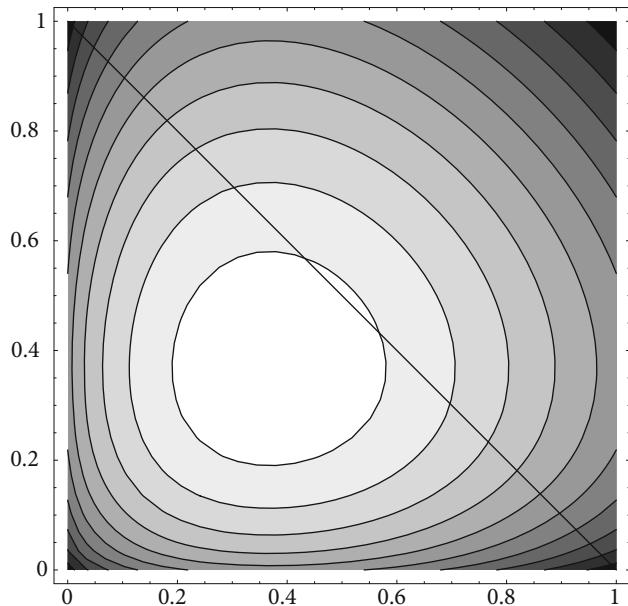
$$p_1 + p_2 + p_3 + p_4 = 1 \tag{7.6}$$

$$(7.4) \text{ in } (7.5): \quad p_2 = \alpha - \alpha\beta = \alpha(1 - \beta) \tag{7.7}$$

$$(7.5) \text{ in } (7.6): \quad p_3 + p_4 = 1 - \alpha \tag{7.8}$$

Die Wahrscheinlichkeiten p_1, p_2 für die Welten (A, B) und $(A, \neg B)$ sind nun also bekannt, aber für die beiden Werte p_3, p_4 bleibt nur noch eine Gleichung. Um trotz des fehlenden Wissens zu einer eindeutigen Lösung zu kommen, wechseln wir den Standpunkt. Wir

Abb. 7.2 Höhenliniendiagramm der zweidimensionalen Entropiefunktion. Man erkennt, dass sie im gesamten Einheitsquadrat streng konvex ist und dass sie ein isoliertes globales Maximum besitzt. Eingezeichnet ist außerdem die Nebenbedingung $p_3 + p_4 = 1$ als Spezialfall der hier relevanten Randbedingung $p_3 + p_4 - 1 + \alpha = 0$ für $\alpha = 0$



verwenden die gegebene Gleichung als Randbedingung zur Lösung eines Optimierungsproblems.

Gesucht ist nun eine Verteilung \mathbf{p} (für die Variablen p_3, p_4), welche die Entropie

$$H(\mathbf{p}) = -\sum_{i=1}^n p_i \ln p_i = -p_3 \ln p_3 - p_4 \ln p_4 \quad (7.9)$$

unter der Nebenbedingung $p_3 + p_4 = 1 - \alpha$ (7.8) maximiert. Warum soll gerade die Entropiefunktion maximiert werden? Da uns hier Wissen über die Verteilung fehlt, muss dieses irgendwie hinzugefügt werden. Wir könnten ad hoc einen Wert, zum Beispiel $p_3 = 0,1$, festsetzen. Besser ist es, die Werte p_3 und p_4 so zu bestimmen, dass die hinzugefügte Information so gering wie möglich ist. Man kann zeigen (Abschn. 8.4.2 und [SW76]), dass die Entropie bis auf einen konstanten Faktor die Unsicherheit einer Verteilung misst. Die negative Entropie ist dann also ein Maß für den Informationsgehalt der Verteilung. Das Maximieren der Entropie minimiert also den Informationsgehalt der Verteilung. Zur Veranschaulichung ist in Abb. 7.2 die Entropiefunktion für den zweidimensionalen Fall grafisch dargestellt.

Zur Bestimmung des Maximums der Entropie unter der Nebenbedingung $p_3 + p_4 - 1 + \alpha = 0$ verwenden wir die Methode der Lagrange parameter [BHW89]. Die Lagrangefunktion lautet

$$L = -p_3 \ln p_3 - p_4 \ln p_4 + \lambda(p_3 + p_4 - 1 + \alpha).$$

Partiell abgeleitet nach p_3 , p_4 und λ erhalten wir

$$\begin{aligned}\frac{\partial L}{\partial p_3} &= -\ln p_3 - 1 + \lambda = 0 \\ \frac{\partial L}{\partial p_4} &= -\ln p_4 - 1 + \lambda = 0\end{aligned}$$

und berechnen

$$p_3 = p_4 = \frac{1-\alpha}{2}.$$

Nun können wir den gesuchten Wert

$$P(B) = P(A, B) + P(\neg A, B) = p_1 + p_3 = \alpha\beta + \frac{1-\alpha}{2} = \alpha\left(\beta - \frac{1}{2}\right) + \frac{1}{2}$$

berechnen. Einsetzen von α und β ergibt

$$P(B) = P(A) \left(P(B|A) - \frac{1}{2} \right) + \frac{1}{2}.$$

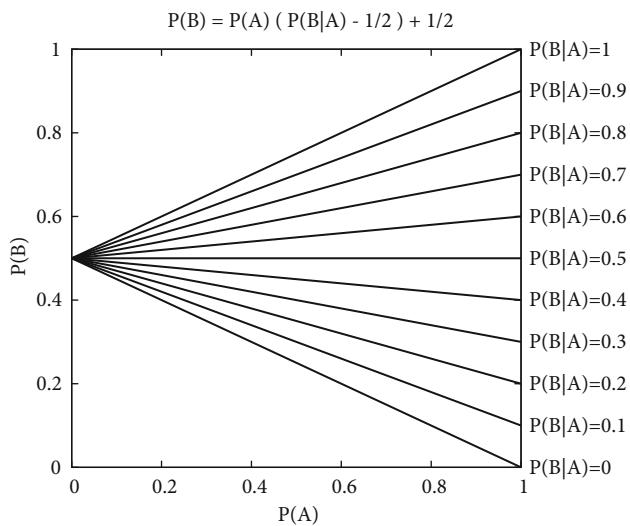
In Abb. 7.3 ist $P(B)$ für verschiedene Werte von $P(B|A)$ dargestellt. Man erkennt, dass im zweiwertigen Grenzfall, das heißt, wenn $P(B)$ und $P(B|A)$ die Werte 0 oder 1 annehmen, die probabilistische Inferenz die gleichen Werte für $P(B)$ liefert wie der Modus Ponens. Wenn A und $B|A$ beide wahr sind, ist auch B wahr. Interessant ist der Fall $P(A) = 0$, in dem $\neg A$ wahr ist. Der Modus Ponens ist hier nicht anwendbar, aber unsere Formel ergibt unabhängig von $P(B|A)$ den Wert $1/2$ für $P(B)$. Wenn A falsch ist, wissen wir nichts über B , was genau unserer Intuition entspricht. Auch der Fall $P(A) = 1$ und $P(B|A) = 0$ deckt sich mit der Aussagenlogik. Hier ist A wahr und $A \rightarrow B$ falsch, also $A \wedge \neg B$ wahr. Also ist B falsch. Die waagrechte Gerade in der Abbildung sagt, dass wir im Fall $P(B|A) = 1/2$ über B nichts aussagen können. Zwischen diesen Punkten verändert sich $P(B)$ linear bei Änderung von $P(A)$ oder $P(B|A)$.

Satz 7.3

Sei eine konsistente² Menge von linearen probabilistischen Gleichungen gegeben. Dann existiert ein eindeutiges Maximum der Entropiefunktion unter den gegebenen Gleichungen als Nebenbedingungen. Die dadurch definierte MaxEnt-Verteilung besitzt unter den Nebenbedingungen minimalen Informationsgehalt.

² Eine Menge von probabilistischen Gleichungen heißt konsistent, wenn es mindestens eine Lösung (das heißt mindestens eine Verteilung) gibt, die alle Gleichungen erfüllt.

Abb. 7.3 Die Kurvenschar für $P(B)$ in Abhängigkeit von $P(A)$ für verschiedene Werte von $P(B|A)$



Aus diesem Satz folgt, dass es keine Verteilung gibt, welche die Nebenbedingungen erfüllt und eine höhere Entropie als die MaxEnt-Verteilung hat. Ein Kalkül, der zu Verteilungen mit geringerer Entropie führt, fügt ad hoc Informationen hinzu, was nicht gerechtfertigt ist.

Bei genauerer Betrachtung der obigen Berechnung von $P(B)$ erkennt man, dass die beiden Werte p_3 und p_4 immer symmetrisch vorkommen. Das heißt, bei Vertauschung der beiden Variablen ändert sich das Ergebnis nicht. Daher ergibt sich am Ende $p_3 = p_4$. Die so genannte Indifferenz dieser beiden Variablen führt dazu, dass MaxEnt sie gleichsetzt. Dieser Zusammenhang gilt allgemein:

Definition 7.5

Wenn eine beliebige Vertauschung von zwei oder mehr Variablen in den Lagrange-Gleichungen diese in einen Satz äquivalenter Gleichungen überführt, so nennt man diese Variablen **indifferent**.

Satz 7.4

Ist eine Menge von Variablen $\{p_{i_1}, \dots, p_{i_k}\}$ indifferent, so liegt das Entropiemaximum unter den gegebenen Nebenbedingungen an einem Punkt mit $p_{i_1} = p_{i_2} = \dots = p_{i_k}$.

Mit diesem Wissen hätten wir bei der Berechnung von $P(B)$ die beiden Variablen p_3 und p_4 (ohne die Lagrange-Gleichungen zu lösen) sofort gleichsetzen können.

7.2.2 Entropimaximum ohne explizite Nebenbedingungen

Wir betrachten nun den Fall, dass gar kein Wissen gegeben ist. Das heißt, es gibt außer der Normierungsbedingung

$$p_1 + p_2 + \dots + p_n = 1$$

keine Nebenbedingungen. Alle Variablen sind daher indifferent. Wir können sie also gleichsetzen und es folgt $p_1 = p_2 = \dots = p_n = 1/n$.³ Für das Schließen mit Unsicherheit bedeutet dies, dass bei völligem Fehlen von Wissen immer alle Welten gleich wahrscheinlich sind. Das heißt, die Verteilung ist eine Gleichverteilung. Im Fall von zwei Variablen A und B würde also gelten

$$P(A, B) = P(A, \neg B) = P(\neg A, B) = P(\neg A, \neg B) = 1/4,$$

woraus zum Beispiel $P(A) = P(B) = 1/2$ und $P(B|A) = 1/2$ folgt. Das Ergebnis für den zweidimensionalen Fall kann man auch an Abb. 7.2 ablesen, denn die eingezeichnete Nebenbedingung ist genau die Normierungsbedingung. Man erkennt, dass das Maximum der Entropie entlang der Geraden genau bei $(1/2, 1/2)$ liegt.

Sobald der Wert einer Nebenbedingung von dem aus der Gleichverteilung abgeleiteten Wert abweicht, verschieben sich die Wahrscheinlichkeiten der Welten. Dies zeigen wir an einem weiteren Beispiel auf. Mit den gleichen Bezeichnungen wie oben nehmen wir an, dass nur

$$P(B|A) = \beta$$

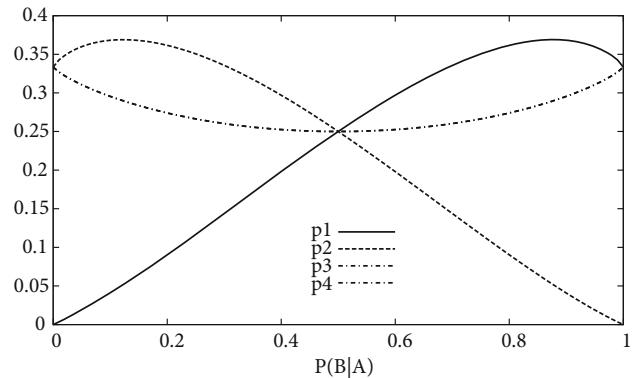
bekannt ist. Also ist $P(A, B) = P(B|A)P(A) = \beta P(A)$, woraus $p_1 = \beta(p_1 + p_2)$ folgt und es ergeben sich die beiden Nebenbedingungen

$$\begin{aligned} \beta p_2 + (\beta - 1)p_1 &= 0 \\ p_1 + p_2 + p_3 + p_4 - 1 &= 0. \end{aligned}$$

Die Lagrangegleichungen lassen sich hier nicht mehr so einfach symbolisch lösen. Numerisches Lösen der Lagrangegleichungen ergibt das in Abb. 7.4 dargestellte Bild, das unter anderem zeigt, dass $p_3 = p_4$. Dies kann man auch schon an den Randbedingungen ablesen, in denen p_3 und p_4 indifferent sind. Für $P(B|A) = 1/2$ erhält man die Gleichverteilung, was nicht überrascht. Das heißt, für diesen Wert bedeutet die Randbedingung keine Einschränkung für die Verteilung. Man erkennt außerdem, dass für kleine $P(B|A)$ auch $P(A, B)$ klein wird.

³ Der Leser möge dieses Resultat durch Maximierung der Entropie unter der Normierungsbedingung berechnen (Aufgabe 7.5).

Abb. 7.4 p_1, p_2, p_3, p_4 in Abhangigkeit von β



7.2.3 Bedingte Wahrscheinlichkeit versus materiale Implikation

Nun zeigen wir, dass die bedingte Wahrscheinlichkeit das intuitive Schlieen besser modelliert als die aus der Logik bekannte materiale Implikation (siehe hierzu auch [Ada75]). Zuerst betrachten wir die in Tab. 7.1 dargestellte Wahrheitstabelle, in der die bedingte Wahrscheinlichkeit und die materiale Implikation fur die Extremfalle der Wahrscheinlichkeiten null und eins verglichen werden. In den beiden intuitiv kritischen Fallen mit falscher Pramisse ist $P(B|A)$ nicht definiert, was Sinn macht.

Nun fragen wir uns, welchen Wert $P(B|A)$ annimmt, wenn beliebige Werte $P(A) = \alpha$ und $P(B) = \gamma$ gegeben sind und sonst kein weiteres Wissen bekannt ist. Wieder maximieren wir die Entropie unter den gegebenen Randbedingungen. Wie oben setzen wir

$$p_1 = P(A, B), \quad p_2 = P(A, \neg B), \quad p_3 = P(\neg A, B), \quad p_4 = P(\neg A, \neg B)$$

und erhalten als Randbedingungen

$$p_1 + p_2 = \alpha \tag{7.10}$$

$$p_1 + p_3 = \gamma \tag{7.11}$$

$$p_1 + p_2 + p_3 + p_4 = 1 \tag{7.12}$$

Damit berechnet man durch Maximierung der Entropie (siehe Aufgabe 7.8)

$$p_1 = \alpha\gamma, \quad p_2 = \alpha(1 - \gamma), \quad p_3 = \gamma(1 - \alpha), \quad p_4 = (1 - \alpha)(1 - \gamma).$$

Aus $p_1 = \alpha\gamma$ folgt $P(A, B) = P(A) \cdot P(B)$, das heit die Unabhangigkeit von A und B . Durch das Fehlen von Randbedingungen, welche die Variablen A und B verknupfen, fuhrt das MaxEnt-Prinzip zur Unabhangigkeit dieser Variablen. Die rechte Halft der Tab. 7.1 lasst sich damit einfacher verstehen. Aus der Definition

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

Tab. 7.1 Wahrheitstabelle für die materiale Implikation und die bedingte Wahrscheinlichkeit für den aussagenlogischen Grenzfall

A	B	$A \Rightarrow B$	$P(A)$	$P(B)$	$P(B A)$
w	w	w	1	1	1
w	f	f	1	0	0
f	w	w	0	1	Nicht definiert
f	f	w	0	0	Nicht definiert

folgt für den Fall $P(A) \neq 0$, das heißtt, wenn die Prämisse nicht falsch ist, wegen der Unabhängigkeit von A und B, dass $P(B|A) = P(B)$. Für den Fall $P(A) = 0$ bleibt $P(B|A)$ undefiniert.

7.2.4 MaxEnt-Systeme

Wie schon erwähnt lässt sich auf Grund der Nichtlinearität der Entropiefunktion die MaxEnt-Optimierung für nicht ganz einfache Probleme meist nicht symbolisch durchführen. Daher wurden für die numerische Maximierung der Entropie zwei Systeme entwickelt. An der Fernuniversität Hagen wurde das System SPIRIT [RM96, BKI00] gebaut und an der Technischen Universität München das System PIT (Probability Induction Tool) [Sch96, ES99, SE00], welches wir nun kurz vorstellen werden.

Das System PIT verwendet die Methode des Sequential Quadratic Programming (SQP), um numerisch ein Extremum der Entropiefunktion unter den gegebenen Nebenbedingungen zu finden. Als Eingabe erwartet PIT eine Datei mit den Randbedingungen. Zum Beispiel die Randbedingungen $P(A) = \alpha$ und $P(B|A) = \beta$ aus Abschn. 7.2.1 haben dann 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]);
```

Wegen der numerischen Berechnung muss man allerdings explizite Wahrscheinlichkeitswerte angeben. Die vorletzte Zeile enthält die Anfrage $QP([B=t])$. Das bedeutet, der Wert $P(B)$ ist gewünscht. Unter <http://www.pit-systems.de> bei „Examples“ gibt man nun diese Eingaben in ein leeres Eingabefenster („Blank Page“) ein und startet PIT. Als Ergebnis erhält man

Nr	Truthvalue	Probability	Query
1	UNSPECIFIED	3.800e-01	$QP([B=t]);$
2	UNSPECIFIED	3.000e-01	$QP([A=t]->[B=t]);$

und liest daraus $P(B) = 0,38$ und $P(B|A) = 0,3$ ab.

7.2.5 Das Tweety-Beispiel

Anhand des Tweety-Beispiels aus Abschn. 4.3 zeigen wir nun, dass das Schließen mit Wahrscheinlichkeiten und insbesondere MaxEnt nicht monoton ist und daher das Alltagsschließen sehr gut modellieren kann. Die relevanten Regeln modellieren wir mit Wahrscheinlichkeiten wie folgt:

$$\begin{aligned} P(\text{Vogel} | \text{Pinguin}) &= 1 && \text{„Pinguine sind Vögel“} \\ P(\text{Fliegt} | \text{Vogel}) &\in [0,95, 1] && \text{„(fast alle) Vögel können fliegen“} \\ P(\text{Fliegt} | \text{Pinguin}) &= 0 && \text{„Pinguine können nicht fliegen“} \end{aligned}$$

Die erste und die dritte Regel stellen harte Aussagen dar, wie sie auch in Logik einfach formuliert werden können. Bei der zweiten hingegen drücken wir unser Wissen, dass fast alle Vögel fliegen können, durch ein Wahrscheinlichkeitsintervall aus. Auf die PIT-Eingabedatei

```
var Pinguin{ja,nein}, Vogel{ja,nein}, Fliegt{ja,nein};
P([Vogel=ja] | [Pinguin=ja]) = 1;
P([Fliegt=ja] | [Vogel=ja]) IN [0.95,1];
P([Fliegt=ja] | [Pinguin=ja]) = 0;

QP([Fliegt=ja] | [Pinguin=ja]);
```

erhalten wir die korrekte Antwort

Nr	Truthvalue	Probability	Query
1	UNSPECIFIED	0.000e+00	QP([Pinguin=ja]->[Fliegt=ja]);

mit der Aussage, dass Pinguine nicht fliegen können.⁴ Die Erklärung hierfür ist ganz einfach. Durch $P(\text{Fliegt} | \text{Vogel}) \in [0,95, 1]$ wird ermöglicht, dass es nicht-fliegende Vögel gibt. Würden wir diese Regel ersetzen durch $P(\text{Fliegt} | \text{Vogel}) = 1$, so hätte auch PIT keine Chance und würde eine Fehlermeldung über inkonsistente Constraints ausgeben.

An diesem Beispiel erkennt man auch sehr gut, dass Wahrscheinlichkeitsintervalle oft sehr hilfreich sind, um unser Unwissen über den exakten Wahrscheinlichkeitswert zu modellieren. Wir hätten die zweite Regel im Sinne von „normalerweise fliegen Vögel“ noch unschärfer formulieren können durch $P(\text{Fliegt} | \text{Vogel}) \in (0,5, 1]$. Die Verwendung des halb offenen Intervalls schließt den Wert 0,5 aus.

Schon in [Pea88] wurde gezeigt, dass sich dieses Beispiel mit Wahrscheinlichkeitslogik, auch ohne MaxEnt, lösen lässt. In [Sch96] wird für eine Reihe von anspruchsvollen

⁴ $\text{QP}([\text{Pinguin}=ja] -> [\text{Fliegt}=ja])$ ist eine alternative Form der PIT-Syntax von $\text{QP}([\text{Fliegt}=ja] | [\text{Pinguin}=ja])$.

Benchmarks für nichtmonotones Schließen aus [Lif89] gezeigt, dass sich diese mit MaxEnt elegant lösen lassen. Eine erfolgreiche praktische Anwendung von MaxEnt in Form eines medizinischen Expertensystems stellen wir im folgenden Abschnitt vor.

7.3 LEXMED, ein Expertensystem für Appendizitisdiagnose

Wir stellen hier das von Manfred Schramm, Walter Rampf und dem Autor an der Hochschule Ravensburg-Weingarten zusammen mit vom Krankenhaus 14-Nothelfer in Weingarten entwickelte medizinische Expertensystem LEXMED vor, das die MaxEnt-Methode verwendet [SE00, Le999].⁵ Die Abkürzung LEXMED steht für *lernfähiges Expertensystem für medizinische Diagnose*.

7.3.1 Appendizitisdiagnose mit formalen Methoden

Die häufigste ernsthafte Ursache für akute Bauchschmerzen [dD91] bildet die Appendizitis, eine Entzündung des Wurmfortsatzes des Blinddarms. Auch heute noch ist die Diagnose in vielen Fällen schwierig [OFY⁺95]. Zum Beispiel sind bis zu ca. 20 % der entfernten Appendix unauffällig, d. h. die entsprechenden Operationen waren unnötig. Ebenso gibt es regelmäßig Fälle, in denen ein entzündeter Appendix nicht als solcher erkannt wird.

Schon seit Anfang der Siebziger Jahre gibt es Bestrebungen, die Appendizitisdiagnose zu automatisieren mit dem Ziel, die Fehldiagnoserate zu verringern [dDLS⁺72, OPB94, OFY⁺95]. Besonders zu erwähnen ist hier das von de Dombal in Großbritannien entwickelte Expertensystem zur Diagnose akuter Bauchschmerzen, das 1972, also deutlich vor dem berühmten System MYCIN, publiziert wurde.

Fast alle in der Medizin bisher verwendeten formalen Diagnoseverfahren basieren auf Scores. Scoresysteme sind denkbar einfach anzuwenden: Für jeden Wert eines Symptoms (zum Beispiel *Fieber* oder *Bauchschmerzen rechts unten*) notiert der Arzt eine bestimmte Anzahl an Punkten. Liegt die Summe der Punkte über einem bestimmten Wert (Schwellwert), wird eine bestimmte Entscheidung vorgeschlagen (z. B. Operation). Bei n Symptomen S_1, \dots, S_n lässt sich ein Score für Appendizitis formal als

$$\text{Diagnose} = \begin{cases} \text{Appendizitis} & \text{falls } w_1 S_1 + \dots + w_n S_n > \Theta \\ \text{negativ,} & \text{sonst} \end{cases}$$

beschreiben. Bei Scores wird also ganz einfach eine Linearkombination von Symptomwerten mit einem Schwellwert Θ verglichen. Die Gewichte der Symptome werden mit statistischen Methoden aus Datenbanken gewonnen. Der Vorteil von Scores ist ihre einfache

⁵ Das Projekt wurde finanziert vom Land Baden-Württemberg im Rahmen der Innovativen Projekte, von der AOK Baden-Württemberg, der Hochschule Ravensburg-Weingarten und vom Krankenhaus 14 Nothelfer in Weingarten.

Anwendbarkeit. Die gewichtete Summe der Punkte lässt sich einfach von Hand berechnen und es wird kein Computer für die Diagnose benötigt.

Aufgrund der Linearität sind Scores aber zu schwach für die Modellierung komplexer Zusammenhänge. Da der Beitrag $w_i S_i$ eines Symptoms S_i zum Score unabhängig von den Werten anderer Symptome berechnet wird, ist klar, dass Scoresysteme keine „Kontexte“ berücksichtigen können. Sie können prinzipiell nicht zwischen Kombinationen der Untersuchungsergebnisse, also z. B. nicht zwischen dem Leukozytenwert eines älteren Patienten und dem eines Jüngeren unterscheiden.

Bei einer fest vorgegebenen Menge von Symptomen sind also bedingte Wahrscheinlichkeitsaussagen deutlich mächtiger als die Scores, da letztere die Abhängigkeiten zwischen verschiedenen Symptomen nicht beschreiben können. Man kann zeigen, dass Scores implizit die Unabhängigkeit aller Symptome voraussetzen.

Bei der Verwendung von Scores tritt noch ein weiteres Problem auf. Um eine gute Diagnosequalität zu erzielen, müssen an die zur statistischen Bestimmung der Gewichte w_i verwendeten Datenbanken hohe Anforderungen gestellt werden. Sie müssen insbesondere repräsentativ sein für die Menge der Patienten im jeweiligen Einsatzbereich des Diagnosesystems. Dies zu gewährleisten ist oft sehr schwierig oder sogar unmöglich. In solchen Fällen sind Scores und andere klassische statistische Methoden nicht oder nur mit hoher Fehlerrate anwendbar.

7.3.2 Hybride Probabilistische Wissensbasis

Mit LEXMED lassen sich komplexe Zusammenhänge, wie sie in der Medizin häufig auftreten, gut modellieren und schnell berechnen. Wesentlich ist dabei die Verwendung von Wahrscheinlichkeitsaussagen, mit denen sich auf intuitive und mathematisch fundierte Weise unsichere und unvollständige Informationen ausdrücken und verarbeiten lassen. Als Beispiel für eine typische Anfrage an das Expertensystem könnte folgende Frage dienen: Wie hoch ist die Wahrscheinlichkeit für einen entzündeten Appendix, wenn der Patient ein 23-jähriger Mann mit Schmerzen im rechten Unterbauch und einem Leukozytenwert von 13.000 ist? Als bedingte Wahrscheinlichkeit formuliert heißt das unter Verwendung der in Tab. 7.2 verwendeten Namen und Wertebereiche für die Symptome

$$P(Bef4 = \text{entzündet} \vee Bef4 = \text{perforiert} \mid \\ Sex2 = \text{männlich} \wedge Alt10 \in 21\text{-}25 \wedge Leuko7 \in 12k\text{-}15k).$$

Durch die Verwendung von Wahrscheinlichkeitsaussagen hat LEXMED die Fähigkeit, auch Informationen aus nicht repräsentativen Datenbanken zu nutzen, da diese Informationen durch andere Quellen geeignet ergänzt werden können. LEXMED liegt eine Datenbank zugrunde, die nur Daten von Patienten enthält, denen der Appendix operativ entfernt wurde. Mit statistischen Methoden werden aus der Datenbank (etwa 400) Regeln generiert, die das in der Datenbank enthaltene Wissen abstrahieren und verfügbar machen [ES99]. Da in dieser Datenbank keine Patienten mit Verdacht auf Appendizitis und negativem, das heißt

Tab. 7.2 Zur Abfrage in LEXMED benutzte Symptome und deren Werte. In der Spalte # ist die Anzahl der Werte des jeweiligen Symptoms angegeben

Symptom	Werte	#	Abk.
Geschlecht	männlich, weiblich	2	Sex2
Alter	0-5, 6-10, 11-15, 16-20, 21-25, 26-35, 36-45, 46-55, 56-65, 65-	10	Alt10
Schmerz 1. Quad.	ja, nein	2	S1Q2
Schmerz 2. Quad.	ja, nein	2	S2Q2
Schmerz 3. Quad.	ja, nein	2	S3Q2
Schmerz 4. Quad.	ja, nein	2	S4Q2
Abwehrspannung	lokal, global, keine	3	Abw3
Loslasschmerz	ja, nein	2	Losl2
S. bei Erschütterung	ja, nein	2	Ersch2
Rektalschmerz	ja, nein	2	RektS2
Darmgeräusche	schwach, normal, vermehrt, keine	4	Darmg4
Pos. Sonographiebef.	ja, nein	2	Sono2
Path. Urinsedim.	ja, nein	2	PathU2
Fieber rektal	-37.3, 37.4-37.6, 37.7-38.0, 38.1-38.4, 38.5-38.9, 39.0-	6	TRek6
Leukozyten	0-6k, 6k-8k, 8k-10k, 10k-12k, 12k-15k, 15k-20k, 20k-	7	Leuko7
Befund	entzündet, perforiert, negativ, andere	4	Bef4

(nicht behandlungsbedürftigem) Befund enthalten sind, muss dieses Wissen aus anderen Quellen hinzugefügt werden.⁶ In LEXMED wurden daher die aus der Datenbank gewonnenen Regeln durch (etwa 100) Regeln von medizinischen Experten und der Fachliteratur ergänzt. Dies führt zu einer hybriden probabilistischen Wissensbasis, welche sowohl aus Daten gewonnenes Wissen als auch explizit von Experten formuliertes Wissen enthält. Da beide Arten von Regeln als bedingte Wahrscheinlichkeiten (siehe zum Beispiel (7.14)) formuliert sind, können sie wie in der Abb. 7.5 dargestellt einfach kombiniert werden.

LEXMED errechnet die Wahrscheinlichkeiten verschiedener Befunde anhand der Wahrscheinlichkeitsverteilung aller relevanten Variablen (siehe Tab. 7.2). Da alle 14 in LEXMED verwendeten Symptome und der Befund als diskrete Variablen modelliert werden (auch stetige Variablen wie der Leukozytenwert werden in Bereiche aufgeteilt), lässt sich die Mächtigkeit der Verteilung (d. h. die Größe des Ereignisraumes) anhand von Tab. 7.2 als das Produkt der Zahl der Werte aller Symptome zu

$$2^{10} \cdot 10 \cdot 3 \cdot 4 \cdot 6 \cdot 7 \cdot 4 = 20.643.840$$

Elementen bestimmen. Aufgrund der Normierungsbedingung aus Satz 7.1 enthält sie also 20.643.839 unabhängige Werte. Jede Regelmenge mit weniger als 20.643.839 Wahrscheinlichkeitswerten beschreibt diesen Ereignisraum eventuell nicht vollständig. Zur Beantwor-

⁶ Dieser negative Befund wird als „unspezifische Bauchschmerzen“, (engl. non specific abdominal pain) oder NSAP bezeichnet.

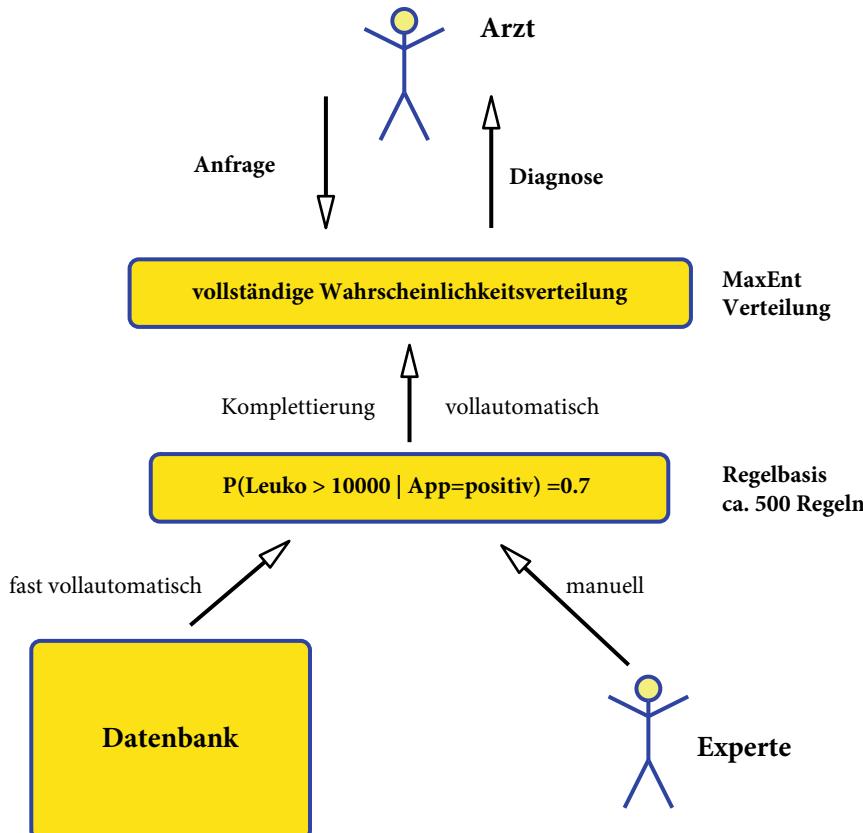


Abb. 7.5 Aus Daten und Expertenwissen werden Wahrscheinlichkeitsregeln generiert, die in einer Regelbasis (Wissensbasis) integriert und anschließend mit der MaxEnt-Methode vervollständigt werden

tung beliebiger Anfragen an das Expertensystem wird aber eine vollständige Verteilung benötigt. Der Aufbau einer so umfangreichen konsistenten Verteilung ist mit statistischen Methoden sehr schwierig.⁷ So gut wie unmöglich wäre es, von einem menschlichen Experten zu verlangen, dass er statt der oben erwähnten 100 Regeln alle 20.643.839 Werte für die Verteilung liefert.

Hier kommt nun die MaxEnt-Methode ins Spiel. Die Generalisierung von etwa 500 Regeln zu einem vollständigen Wahrscheinlichkeitsmodell erfolgt in LEXMED durch Maximierung der Entropie mit den 500 Regeln als Nebenbedingungen. Durch die effiziente Speicherung der resultierenden MaxEnt-Verteilung können die Antwortzeiten für eine Diagnose im Sekundenbereich gehalten werden.

⁷ Die Aufgabe, aus einer Menge von Daten eine Funktion zu generieren, wird als maschinelles Lernen bezeichnet. Hierauf wird in Kap. 8 ausführlich eingegangen.

Personenangaben		nicht bekannt	Werte			
Geschlecht	<input type="radio"/>	<input checked="" type="radio"/> männlich <input type="radio"/> weiblich				<input type="button" value="?"/>
Altersgruppe	<input type="radio"/>	<input type="radio"/> 0-5 <input type="radio"/> 6-10 <input type="radio"/> 11-15 <input type="radio"/> 16-20 <input checked="" type="radio"/> 21-25 <input type="radio"/> 26-35 <input type="radio"/> 36-45 <input type="radio"/> 46-55 <input type="radio"/> 56-65 <input type="radio"/> 65-				<input type="button" value="?"/>
Untersuchungsergebnisse		nicht untersucht	Werte			
1. Schmerzquadrant	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
2. Schmerzquadrant	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
3. Schmerzquadrant	<input type="radio"/>	<input checked="" type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
4. Schmerzquadrant	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Abwehrspannung	<input type="radio"/>	<input checked="" type="radio"/> lokal <input type="radio"/> global <input type="radio"/> keine				<input type="button" value="?"/>
Loslassschmerz	<input type="radio"/>	<input checked="" type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Erschütterungsschmerz	<input type="radio"/>	<input checked="" type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Rektalschmerz	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Darmgeräusche	<input type="radio"/>	<input type="radio"/> schwach <input checked="" type="radio"/> normal <input type="radio"/> vermehrt <input type="radio"/> keine				<input type="button" value="?"/>
Sonographisch auffällig	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Pathologisches Urinsediment	<input checked="" type="radio"/>	<input type="radio"/> ja <input type="radio"/> nein				<input type="button" value="?"/>
Rektaler Temperaturbereich	<input type="radio"/>	<input type="radio"/> -37.3 <input type="radio"/> 37.4-37.6 <input type="radio"/> 37.7-38.0 <input type="radio"/> 38.1-38.4 <input checked="" type="radio"/> 38.5-38.9 <input type="radio"/> 39.0-				<input type="button" value="?"/>
Leukozytenbereich	<input type="radio"/>	<input type="radio"/> 0-6k <input type="radio"/> 6k-8k <input type="radio"/> 8k-10k <input type="radio"/> 10k-12k <input checked="" type="radio"/> 12k-15k <input type="radio"/> 15k-20k <input type="radio"/> 20k-				<input type="button" value="?"/>
Abfragen						
<input type="button" value="Diagnose(4w)"/>	<input type="button" value="?"/>	<input type="button" value="Diagnose(3w)"/>	<input type="button" value="?"/>	<input type="button" value="Datenbankabfrage"/>	<input type="button" value="?"/>	

Ergebnis der PIT-Diagnose				
Diagnose	App. entzündet	App. perforiert	negativ	andere
Wahrscheinlichkeit	0.70	0.17	0.06	0.07

Abb. 7.6 Die LEXMED-Eingabemaske zur Eingabe der untersuchten Symptome und die Ausgabe der Wahrscheinlichkeiten

7.3.3 Anwendung von LEXMED

Die Benutzung von LEXMED ist einfach und selbsterklärend. Der Arzt wählt per Internet die Lexmed-Homepage unter <http://www.lexmed.de> an.⁸ Für eine automatische Diagnose gibt der Arzt die Ergebnisse seiner Untersuchung in die Eingabemaske in Abb. 7.6 ein. Nach 1–2 Sekunden erhält er als Antwort die Wahrscheinlichkeiten für die vier verschie-

⁸ Eine Version mit eingeschränkter Funktionalität ist ohne Passwort zugänglich.

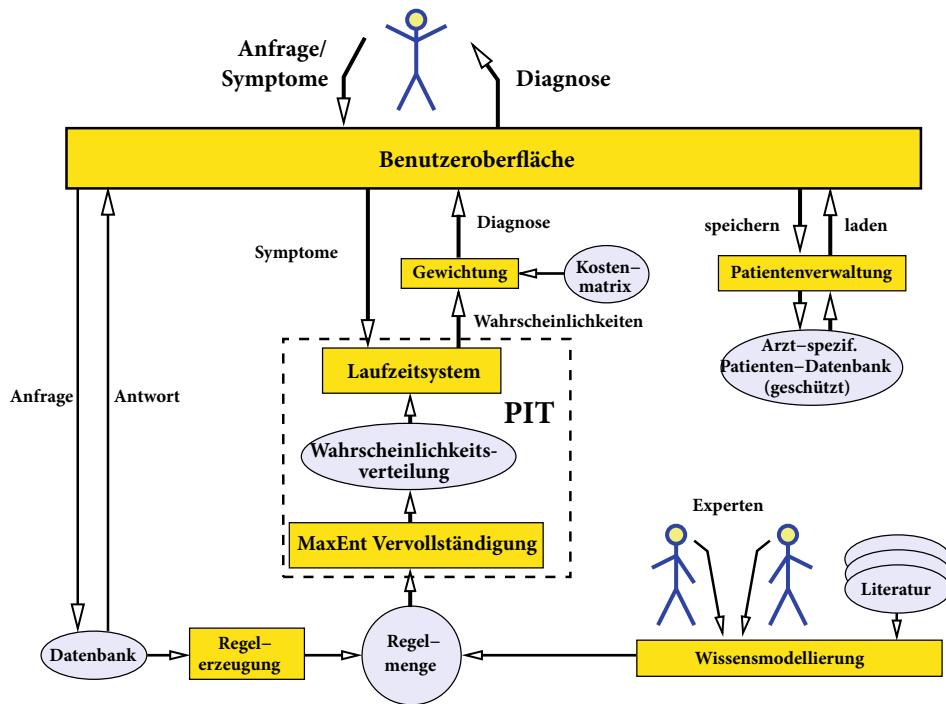


Abb. 7.7 Aus der Datenbank sowie aus Expertenwissen werden Regeln generiert. MaxEnt erzeugt daraus eine vollständige Wahrscheinlichkeitsverteilung. Auf eine Anfrage des Benutzers wird für jede mögliche Diagnose eine Wahrscheinlichkeit berechnet. Mit Hilfe der Kostenmatrix (siehe Abschn. 7.3.5) wird dann eine Entscheidung vorgeschlagen

denen Befunde sowie einen Diagnosevorschlag (Abschn. 7.3.5). Fehlen bei der Eingabe einzelne Untersuchungsergebnisse (z. B. der Sonographiebefund), so wählt der Arzt den Eintrag *nicht untersucht*. Dabei ist die Sicherheit der Diagnose natürlich um so höher, je mehr Symptomwerte eingegeben wurden.

Um seine eingegebenen Daten zu archivieren, verfügt jeder registrierte Benutzer über eine eigene Patientendatenbank, auf die nur er selbst Zugang hat. So können zum Beispiel die Daten und die Diagnosen früherer Patienten einfach mit denen eines neuen Patienten verglichen werden. Eine Gesamtübersicht der Abläufe in LEXMED ist in Abb. 7.7 dargestellt.

7.3.4 Funktion von LEXMED

Das Wissen wird formalisiert mit Hilfe von Wahrscheinlichkeitsaussagen. Beispielsweise legt die Aussage

$$P(\text{Leuko7} > 20.000 \mid \text{Bef4} = \text{entzündet}) = 0,09$$

die Häufigkeit für einen Leukozytenwert von mehr als 20.000 bei einem Befund *entzündet* auf einen Wert von 9 % fest.⁹

Lernen von Regeln durch Statistische Induktion

Die in LEXMED verwendete Datenbank enthält in der Rohfassung 54 verschiedene Werte (anonymisiert) von 14.646 Patienten. Wie schon erwähnt sind in dieser Datenbank nur die Patienten erfasst, denen der Appendix operativ entfernt wurde. Von den in der Datenbank verwendeten 54 Attributen wurden nach einer statistischen Analyse die in Tab. 7.2 dargestellten 14 Symptome verwendet. Aus dieser Datenbank werden nun in zwei Schritten die Regeln erzeugt. Der erste Schritt bestimmt dabei die Abhängigkeitsstruktur der Symptome, der zweite Schritt füllt diese Struktur mit den entsprechenden Wahrscheinlichkeitsregeln.¹⁰

Bestimmung des Abhängigkeitsgraphen Der Graph in Abb. 7.8 enthält für jede Variable (die Symptome und den Befund) 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 von zwei unabhängigen Variablen ist gleich null. Es wurde für jedes der 14 Symptome die Paarkorrelation mit *Bef4* berechnet und in den Graphen eingetragen. Außerdem wurden alle Dreierkorrelationen von Befund mit je zwei Symptomen berechnet. Von diesen wurden nur die stärksten Werte in Form von zusätzlichen Kanten zwischen den zwei beteiligten Symptomen eingezeichnet.

Schätzen der Regelwahrscheinlichkeiten Die Struktur des Abhängigkeitsgraphen beschreibt die Struktur der gelernten Regeln (wie auch bei einem Bayes-Netz üblich¹¹). Die Regeln haben dabei unterschiedliche Komplexität: Es gibt Regeln, die nur die Verteilung der möglichen Befunde beschreiben (A-priori-Regeln, z. B. (7.13)), Regeln, die die Abhängigkeit zwischen dem Befund und einem Symptom beschreiben (Regeln mit einfacher Bedingung, z. B. (7.14)) und schließlich Regeln, die die Abhängigkeit zwischen dem Befund und 2 Symptomen beschreiben, wie sie in Abb. 7.9 in der PIR-Syntax angegeben sind.

$$P(\text{Bef4} = \text{entzündet}) = 0,40 \quad (7.13)$$

$$P(\text{Sono2} = \text{ja} \mid \text{Bef4} = \text{entzündet}) = 0,43 \quad (7.14)$$

$$P(\text{S4Q2} = \text{ja} \mid \text{Bef4} = \text{entzündet} \wedge \text{S2Q2} = \text{ja}) = 0,61 \quad (7.15)$$

⁹ Statt einzelner numerischer Werte können hier auch Intervalle (beispielsweise [0,06, 0,12]) verwendet werden.

¹⁰ Für eine systematische Einführung in das maschinelle Lernen verweisen wir auf Kap. 8.

¹¹ Der Unterschied zu einem Bayesnetz liegt z. B. darin, dass die Regeln noch mit Wahrscheinlichkeitsintervallen versehen sind und erst nach Anwendung des Prinzips maximaler Entropie ein eindeutiges Wahrscheinlichkeitsmodell erzeugt wird.

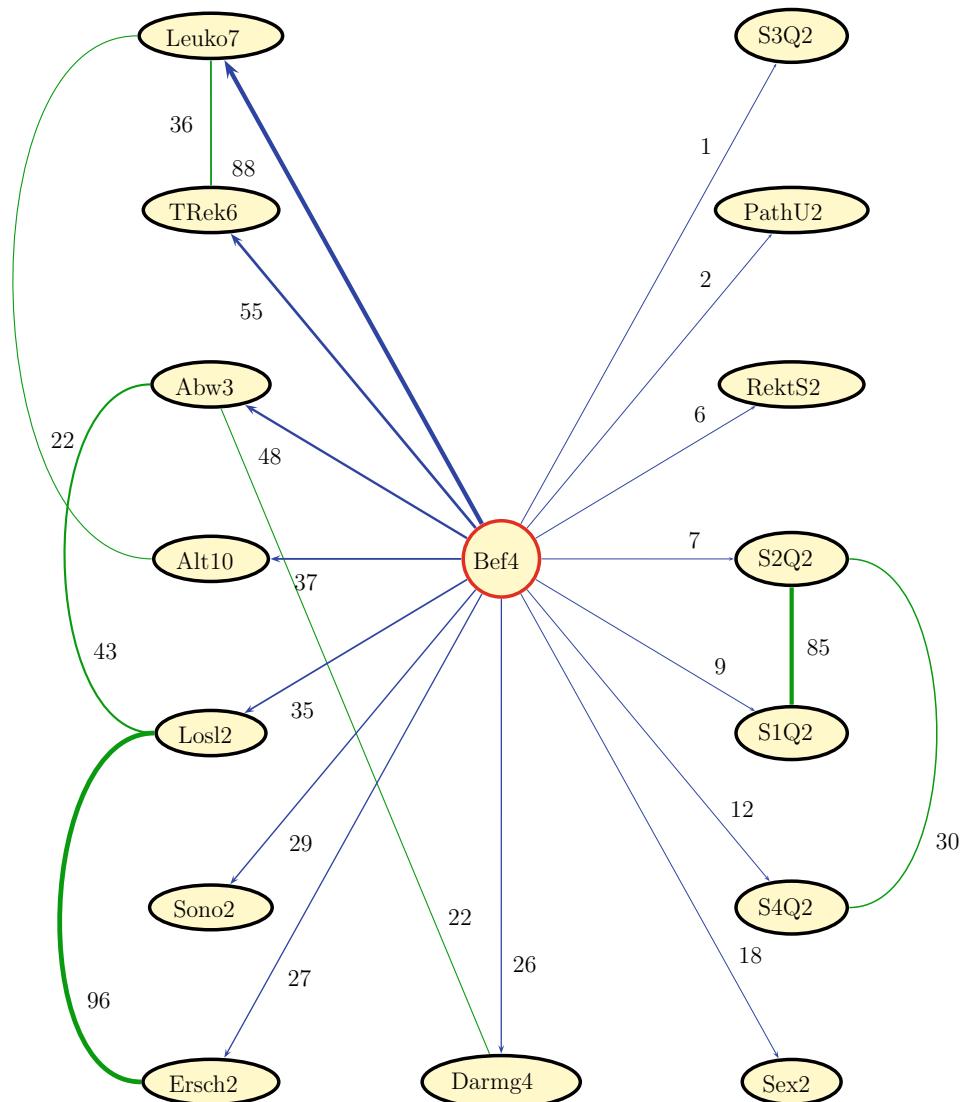


Abb. 7.8 Der aus der Datenbank berechnete Abhängigkeitsgraph

Um die Kontextabhängigkeit des gespeicherten Wissens so gering wie möglich zu halten, enthalten alle Regeln den Befund in ihrer Bedingung und nicht als Konklusion. Dies ist ganz analog zum Aufbau vieler Medizinbücher mit Formulierungen der Art „Bei Appendizitis finden wir gewöhnlich ...“. Wie schon in Beispiel 7.6 gezeigt wurde, stellt dies jedoch kein Problem dar, denn mit Hilfe der Bayes-Formel bringt LEXMED diese Regeln automatisch in die richtige Form.

```

1 P([Leuko7=0-6k] | [Bef4=negativ] * [Alt10=16-20]) = [0.132,0.156];
2 P([Leuko7=6-8k] | [Bef4=negativ] * [Alt10=16-20]) = [0.257,0.281];
3 P([Leuko7=8-10k] | [Bef4=negativ] * [Alt10=16-20]) = [0.250,0.274];
4 P([Leuko7=10-12k] | [Bef4=negativ] * [Alt10=16-20]) = [0.159,0.183];
5 P([Leuko7=12-15k] | [Bef4=negativ] * [Alt10=16-20]) = [0.087,0.112];
6 P([Leuko7=15-20k] | [Bef4=negativ] * [Alt10=16-20]) = [0.032,0.056];
7 P([Leuko7=20k-] | [Bef4=negativ] * [Alt10=16-20]) = [0.000,0.023];
8 P([Leuko7=0-6k] | [Bef4=negativ] * [Alt10=21-25]) = [0.132,0.172];
9 P([Leuko7=6-8k] | [Bef4=negativ] * [Alt10=21-25]) = [0.227,0.266];
10 P([Leuko7=8-10k] | [Bef4=negativ] * [Alt10=21-25]) = [0.211,0.250];
11 P([Leuko7=10-12k] | [Bef4=negativ] * [Alt10=21-25]) = [0.166,0.205];
12 P([Leuko7=12-15k] | [Bef4=negativ] * [Alt10=21-25]) = [0.081,0.120];
13 P([Leuko7=15-20k] | [Bef4=negativ] * [Alt10=21-25]) = [0.041,0.081];
14 P([Leuko7=20k-] | [Bef4=negativ] * [Alt10=21-25]) = [0.004,0.043];

```

Abb. 7.9 Einige der LEXMED-Regeln mit Wahrscheinlichkeitsintervallen. „*“ steht hier für „ \wedge “

Die numerischen Werte für diese Regeln werden durch Zählen der Häufigkeit in der Datenbank geschätzt. Zum Beispiel ergibt sich der Wert in (7.14) durch Abzählen und Berechnen von

$$\frac{|Bef4 = entzündet \wedge Sono2 = ja|}{|Bef4 = entzündet|} = 0,43.$$

Expertenregeln

Da die Appendektomie-Datenbank nur die operierten Patienten enthält, werden die Regeln für unspezifische Bauchschmerzen (NSAP) durch Wahrscheinlichkeitsaussagen von medizinischen Experten beschrieben. Die Erfahrungen in LEXMED bestätigen, dass die verwendeten Wahrscheinlichkeitsaussagen einfach zu lesen und direkt in die Umgangssprache zu übersetzen sind. Aussagen von medizinischen Experten über Häufigkeitsbeziehungen bestimmter Symptome und Befunde, sei es aus der Literatur oder als Ergebnis einer Befragung, konnten daher ohne großen Aufwand in die Regelbasis aufgenommen werden. Zur Modellierung der Unsicherheit des Expertenwissens hat sich hier die Verwendung von Wahrscheinlichkeitsintervallen bewährt. Das Expertenwissen wurde im Wesentlichen von dem beteiligten Chirurgen, Dr. Rampf, und von Dr. Hontschik sowie seinen Publikationen [Hon94] akquiriert.

Sind die Expertenregeln erzeugt, ist die Regelbasis fertig. Mit der Methode der maximalen Entropie wird nun das vollständige Wahrscheinlichkeitsmodell berechnet.

Diagnoseanfragen

Aus dem effizient abgespeicherten Wahrscheinlichkeitsmodell berechnet LEXMED innerhalb weniger Sekunden aus den eingegebenen Symptomwerten die Wahrscheinlichkeiten für die vier möglichen Befunde. Als Beispiel nehmen wir folgende Ausgabe an:

Ergebnis der PIT-Diagnose				
Diagnose	App. entzündet	App. perforiert	negativ	andere
Wahrscheinlichkeit	0,24	0,16	0,57	0,03

Basierend auf diesen vier Wahrscheinlichkeitswerten muss eine Entscheidung für eine der vier Therapien Operation, Notoperation, stationär beobachten oder ambulant beobachten¹² erfolgen. Obwohl hier die Wahrscheinlichkeit für einen negativen Befund überwiegt, ist es keine gute Entscheidung, den Patienten als gesund nach Hause zu schicken. Man erkennt deutlich, dass die Diagnose mit der Berechnung der Wahrscheinlichkeiten für die Befunde noch nicht abgeschlossen ist.

Vielmehr steht nun die Aufgabe an, aus diesen Wahrscheinlichkeiten eine optimale Entscheidung abzuleiten. Hierzu kann sich der Benutzer von LEXMED einen Entscheidungsvorschlag berechnen lassen.

7.3.5 Risikomanagement mit Hilfe der Kostenmatrix

Wie können nun die berechneten Wahrscheinlichkeiten optimal in Entscheidungen überetzt werden? Ein naives Verfahren würde jedem Befund eine Entscheidung zuordnen und anschließend diejenige Entscheidung wählen, die dem Befund mit der höchsten Wahrscheinlichkeit entspricht. Angenommen die berechneten Wahrscheinlichkeiten sind 0,40 für den Befund *Appendizitis* (entzündet oder perforiert), 0,55 für den Befund *negativ* und 0,05 für den Befund *andere*. Ein naives Verfahren würde nun die (offensichtlich bedenkliche) Entscheidung „keine Operation“ wählen, da sie zu dem Befund mit der größeren Wahrscheinlichkeit korrespondiert. Eine bessere Methode besteht darin, die Kosten der möglichen Fehler zu vergleichen, die bei den jeweiligen Entscheidungen auftreten können. Der Fehler wird in der Form von „(theoretischen) Mehrkosten der aktuellen gegenüber der optimalen Entscheidung“ quantifiziert. Die angegebenen Werte enthalten Kosten des Krankenhauses, der Krankenkasse, des Patienten und anderer Parteien (z. B. Arbeitsausfall), unter Einbeziehung der Spätfolgen. Diese Kosten sind in Tab. 7.3 angegeben.

Die Beträge werden anschließend für jede Entscheidung gemittelt, d. h. unter Berücksichtigung ihrer Häufigkeiten summiert. Diese sind in der letzten Spalte in Tab. 7.3 eingetragen. Anschließend wird die Entscheidung mit den geringsten mittleren Fehlerkosten vorgeschlagen. In Tab. 7.3 ist die Matrix gemeinsam mit dem für einen Patienten ermittelten Wahrscheinlichkeitsvektor (hier: (0,25, 0,15, 0,55, 0,05)) angegeben. Die letzte Spalte der Tabelle enthält das Ergebnis der Berechnungen der durchschnittlich zu erwartenden Kosten für Fehlentscheidungen. Der Wert für *Operation* in der ersten Zeile berechnet sich daher als gewichtetes Mittel aller Kosten zu $0,25 \cdot 0 + 0,15 \cdot 500 + 0,55 \cdot 5800 + 0,05 \cdot 6000 = 3565$. Optimale Entscheidungen sind mit (Mehr-)Kosten 0 eingetragen. Das System entscheidet sich nun für die Therapie mit den minimalen mittleren Kosten. Es stellt damit einen kostenorientierten Agenten dar.

¹² Ambulant beobachten bedeutet, dass der Patient nach Hause entlassen wird.

Tab. 7.3 Die Kostenmatrix von LEXMED zusammen mit den berechneten Befundwahrscheinlichkeiten eines Patienten

	Wahrscheinlichkeit für versch. Befunde			
	entzündet	perforiert	negativ	andere
Therapie	0,25	0,15	0,55	0,05
Operation	0	500	5800	6000
Not-Operation	500	0	6300	6500
Ambulant beob.	12.000	150.000	0	16.500
Sonstiges	3000	5000	1300	0
Stationär beob.	3500	7000	400	600
				2175

Kostenmatrix im binären Fall

Zum besseren Verständnis von Kostenmatrix und Risikomanagement wollen wir nun das LEXMED-System auf die zweiwertige Unterscheidung zwischen den Befunden *Appendizitis* und *NSAP* einschränken. Als mögliche Therapie kann hier nur *Operation* mit Wahrscheinlichkeit p_1 und *Ambulant beobachten* mit Wahrscheinlichkeit p_2 gewählt werden. Die Kostenmatrix ist dann also eine 2×2 -Matrix der Gestalt

$$\begin{pmatrix} 0 & k_2 \\ k_1 & 0 \end{pmatrix}.$$

Die beiden Nullen in der Diagonalen stehen für die korrekten Entscheidungen *Operation* im Falle von *Appendizitis* und *Ambulant beobachten* bei *NSAP*. Der Parameter k_2 steht für die zu erwartenden Kosten, die anfallen, wenn ein Patient ohne entzündeten Appendix operiert wird. Diese Fehlentscheidung wird als **falsch positiv** bezeichnet. **Falsch negativ** hingegen ist die Entscheidung *ambulant beobachten* im Fall einer Appendizitis. Der Wahrscheinlichkeitsvektor $(p_1, p_2)^T$ wird nun mit dieser Matrix multipliziert und man erhält den Vektor

$$(k_2 p_2, k_1 p_1)^T$$

mit den mittleren Mehrkosten für die beiden möglichen Therapien. Da für die Entscheidung nur das Verhältnis aus den beiden Komponenten relevant ist, kann der Vektor mit einem Faktor multipliziert werden. Wir wählen $1/k_1$ und erhalten $((k_2/k_1)p_2, p_1)$. Relevant ist hier also nur das Verhältnis $k = k_2/k_1$. Das gleiche Ergebnis liefert auch die einfachere Kostenmatrix

$$\begin{pmatrix} 0 & k \\ 1 & 0 \end{pmatrix},$$

welche nur den einen variablen Parameter k enthält. Dieser Parameter ist sehr wichtig, denn er bestimmt das Risikomanagement. Durch Veränderung von k kann der „Arbeitspunkt“ des Diagnosesystems angepasst werden. Für $k \rightarrow \infty$ ist das System extrem riskant

eingestellt, denn es wird kein Patient operiert; mit der Konsequenz, dass es keine falsch positiven Klassifikationen gibt, aber viele falsch negative. Genau umgekehrt sind die Verhältnisse im Fall $k = 0$, denn hier werden alle Patienten operiert.

7.3.6 Leistungsfähigkeit

LEXMED ist vorgesehen für den Einsatz in einer Praxis oder Ambulanz. Vorbedingung für die Anwendung von LEXMED sind akute Bauchschmerzen seit mehreren Stunden (aber weniger als 5 Tage). Weiter ist LEXMED (bisher) spezialisiert auf die Unterscheidung Appendizitis-keine Appendizitis, d.h. für andere Krankheiten enthält das System erst wenig Informationen.

Im Rahmen einer prospektiven Studie wurde im Krankenhaus 14 Nothelfer von Juni 1999 bis Oktober 2000 eine repräsentative Datenbank mit 185 Fällen erstellt. Sie enthält diejenigen Patienten des Krankenhauses, die nach mehreren Stunden akuter Bauchschmerzen mit Verdacht auf Appendizitis in die Klinik kamen. Von diesen Patienten wurden die Symptome und der (im Falle einer Operation histologisch gesicherte) Befund notiert.

Wurden die Patienten nach einigen Stunden oder 1–2 Tagen stationären Aufenthalts (ohne Operation) ganz oder fast beschwerdefrei nach Hause entlassen, wurde telefonisch nachgefragt, ob die Patienten beschwerdefrei geblieben waren oder bei einer nachfolgenden Behandlung ein positiver Befund bestätigt werden konnte.

Um die Darstellung der Ergebnisse zu vereinfachen und eine bessere Vergleichbarkeit mit ähnlichen Studien zu erreichen, wurde LEXMED, wie in Abschn. 7.3.5 beschrieben, auf die zweiwertige Unterscheidung zwischen den Befunden *Appendizitis* und NSAP eingeschränkt. Nun wird k variiert zwischen null und unendlich und für jeden Wert von k werden **Sensitivität** und **Spezifität** auf den Testdaten gemessen. Die Sensitivität misst

$$P(\text{positiv klassifiziert} \mid \text{positiv}) = \frac{|\text{positiv und positiv klassifiziert}|}{|\text{positiv}|},$$

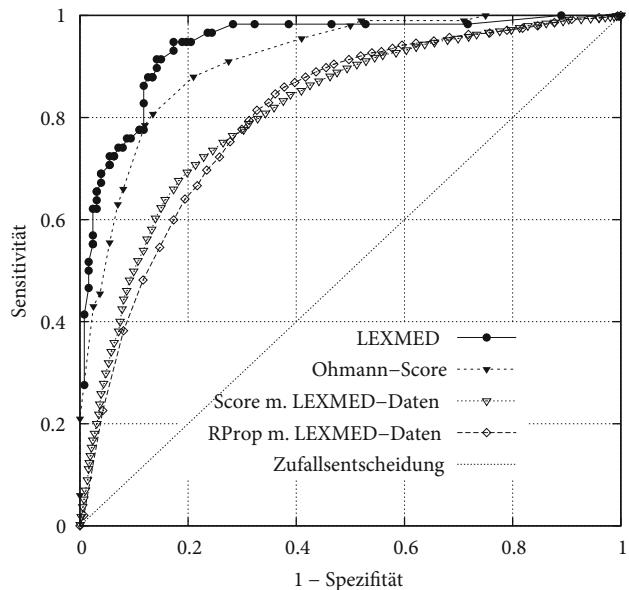
das heißt, den relativen Anteil der positiven Fälle, die korrekt erkannt werden. Sie gibt an, wie sensitiv das Diagnosesystem ist. Die Spezifität hingegen misst

$$P(\text{negativ klassifiziert} \mid \text{negativ}) = \frac{|\text{negativ und negativ klassifiziert}|}{|\text{negativ}|},$$

das heißt, den relativen Anteil der negativen Fälle, die korrekt erkannt werden.

In Abhängigkeit von $0 \leq k < \infty$ geben wir in Abb. 7.10 die Ergebnisse der Sensitivität und Spezifität an. Diese Kurve wird als **ROC-Kurve**, beziehungsweise Receiver Operating Characteristic, bezeichnet. Bevor wir zur Analyse der Qualität von LEXMED kommen, ein paar Worte zur Bedeutung der ROC-Kurve. Zur Orientierung wurde in das Diagramm die Winkelhalbierende eingezeichnet. Alle Punkte auf dieser entsprechen Zufallsentscheidungen. Zum Beispiel der Punkt $(0,2,0,2)$ entspricht einer Spezifität von 0,8 bei einer

Abb. 7.10 ROC-Kurve von LEXMED verglichen mit dem Ohmann-Score und zwei weiteren Modellen



Sensitivität von 0,2. Dies erreicht man ganz einfach, indem man einen neuen Fall, ohne ihn anzusehen, mit einer Wahrscheinlichkeit von 0,2 als positiv und 0,8 als negativ klassifiziert. Jedes wissensbasierte Diagnosesystem muss daher eine ROC-Kurve erzeugen, die deutlich über der Winkelhalbierenden 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 sich bei jedem Patienten für die Operation entscheidet und damit eine Sensitivität von 1 erreicht. Man könnte die ROC-Kurve auch als Kennlinie für zweiwertige Diagnosesysteme bezeichnen. Das ideale Diagnosesystem hätte eine Kennlinie, die letztlich nur noch aus dem Punkt (0,1) besteht, also 100 % Sensitivität und 100 % Spezifität.

Nun zur Analyse der ROC-Kurve. Bei einer Sensitivität von 88 % erreicht LEXMED eine Spezifität von 87 % ($k = 0,6$). Zum Vergleich ist der Ohmann-Score, ein etablierter, bekannter Score für Appendizitis eingetragen [OMYL96, ZSR⁺99]. Da LEXMED fast überall über dem Ohmann-Score beziehungsweise links davon liegt, ist seine mittlere Diagnosequalität deutlich besser. Dies ist nicht überraschend, denn Scores sind einfach zu schwach, um interessante Aussagen zu modellieren. In Abschn. 8.7, beziehungsweise in Aufgabe 8.17 werden wir zeigen, dass die Scores äquivalent sind zum Spezialfall Naive-Bayes, das heißt zur Annahme, alle Symptome sind paarweise unabhängig, wenn die Diagnose bekannt ist. Zum Vergleich von LEXMED mit Scores sollte aber auch erwähnt werden, dass für den Ohmann-Score eine statistisch repräsentative Datenbank verwendet wurde, für LEXMED hingegen eine nicht repräsentative Datenbank, ergänzt durch Expertenwissen. Um eine Vorstellung von der Qualität der LEXMED-Daten im Vergleich zu den Ohmann-Daten zu bekommen,

wurde mit der Methode der kleinsten Quadrate (siehe Abschn. 9.4.1) auf den LEXMED-Daten ein linearer Score berechnet, der zum Vergleich auch noch mit eingezeichnet ist. Außerdem wurde mit dem RProp-Verfahren ein neuronales Netz auf den LEXMED-Daten trainiert (siehe Abschn. 9.5). Man erkennt an der Differenz zwischen der LEXMED-Kurve und dem Score sowie RProp, wie stark die in LEXMED verwendete Methode der Kombination von Daten und Expertenwissen ist.

7.3.7 Einsatzgebiete und Erfahrungen

LEXMED kann und soll das Urteil eines erfahrenen Chirurgen nicht ersetzen. Da jedoch ein Spezialist selbst in klinischen Einrichtungen nicht immer verfügbar ist, bietet sich eine LEXMED-Anfrage als begründete Zweitmeinung an. Besonders interessant und lohnend ist daher der Einsatz in der klinischen Ambulanz und beim niedergelassenen Arzt.

Die Lernfähigkeit von LEXMED, welche die Berücksichtigung weiterer Symptome, weiterer Patientendaten und weiterer Regeln ermöglicht, bietet aber auch neue Möglichkeiten in der Klinik: Bei besonders seltenen und diagnostisch schwierigen Gruppen, zum Beispiel Kinder unter 6 Jahren, kann LEXMED durch die Verwendung der Daten von Kinderärzten oder anderer spezieller Datenbanken auch dem erfahrenen Chirurgen eine Unterstützung liefern.

Neben dem direkten Einsatz in der Diagnostik unterstützt LEXMED auch Maßnahmen der Qualitätssicherung. Zum Beispiel könnten die Krankenkassen die Diagnosequalität von Krankenhäusern mit der von Expertensystemen vergleichen. Durch die Weiterentwicklung der in LEXMED erstellten Kostenmatrix (im Konsens mit Ärzten, Kassen und Patienten) wird die Qualität von ärztlichen Diagnosen, Computerdiagnosen und anderen medizinischen Einrichtungen besser vergleichbar.

Mit LEXMED wurde ein neuer Weg zur Konstruktion von automatischen Diagnosesystemen aufgezeigt. Mit Hilfe der Sprache der Wahrscheinlichkeitstheorie und dem Max-Ent-Verfahren wird induktiv statistisch abgeleitetes Wissen kombiniert mit Wissen von Experten und aus der Literatur. Der auf Wahrscheinlichkeitsmodellen basierende Ansatz ist theoretisch elegant, allgemein anwendbar und liefert in einer kleinen Studie sehr gute Ergebnisse.

LEXMED ist seit 1999 im praktischen Einsatz im 14-Nothelfer-Krankenhaus in Weingarten und hat sich dort sehr gut bewährt. Es steht außerdem unter <http://www.lexmed.de>, ohne Gewähr natürlich, im Internet für jeden Arzt zur freien Verfügung bereit. Seine Diagnosequalität ist vergleichbar mit der eines erfahrenen Chirurgen und damit besser als die eines durchschnittlichen niedergelassenen Arztes, beziehungsweise die eines unerfahrenen Arztes in der Klinik.

Trotz dieses Erfolges zeigt es sich, dass es im deutschen Gesundheitswesen sehr schwierig ist, solch ein System kommerziell zu vermarkten. Ein Grund hierfür ist der fehlende freie Markt, der durch seine Selektionsmechanismen bessere Qualität (hier bessere Diagnose) fördert. Außerdem ist wohl in der Medizin der Zeitpunkt für den breiten Einsatz intelligenter Techniken im Jahr 2007 immer noch nicht gekommen. Eine Ursache hierfür

könnte eine in dieser Hinsicht konservative Lehre an manchen deutschen Medizinfakultäten sein.

Ein weiterer Aspekt ist der Wunsch vieler Patienten nach persönlicher Beratung und Betreuung durch den Arzt, verbunden mit der Angst, dass mit der Einführung von Expertensystemen der Patient nur noch mit dem Automaten kommuniziert. Diese Angst ist jedoch völlig unbegründet. Auch langfristig werden medizinische Expertensysteme den Arzt nicht ersetzen können. Sie könnten aber schon heute, genauso wie Lasermedizin und Kernspintomographie, gewinnbringend für alle Beteiligten eingesetzt werden. Seit dem ersten medizinischen Computerdiagnosesystem von de Dombal 1972 sind nun über 40 Jahre vergangen. Es bleibt abzuwarten, ob es in Deutschland noch weitere 40 Jahre dauern wird, bis die Computerdiagnose zum etablierten medizinischen Handwerkszeug gehört.

7.4 Schließen mit Bayes-Netzen

Ein Problem beim Schließen mit Wahrscheinlichkeiten in der Praxis wurde schon in Abschn. 7.1 aufgezeigt. Wenn d Variablen X_1, \dots, X_d mit je n Werten verwendet werden, so enthält die zugehörige Wahrscheinlichkeitsverteilung insgesamt n^d Werte. Das heißt, dass im Worst-Case der benötigte Speicherplatz und die Rechenzeit für die Erfassung der Verteilung sowie die Rechenzeit für die Bestimmung bestimmter Wahrscheinlichkeiten exponentiell mit der Zahl der Variablen wächst.

In der Praxis sind die Anwendungen aber meist stark strukturiert und die Verteilung enthält viel Redundanz. Das heißt, sie lässt sich mit geeigneten Methoden stark reduzieren. Die Verwendung von so genannten Bayes-Netzen hat sich hier bestens bewährt und gehört heute zu den erfolgreich in der Praxis eingesetzten KI-Techniken. Bei Bayes-Netzen wird Wissen über die Unabhängigkeit von Variablen zur Vereinfachung des Modells verwendet.

7.4.1 Unabhängige Variablen

Im einfachsten Fall sind alle Variablen paarweise unabhängig und es gilt

$$\mathbf{P}(X_1, \dots, X_d) = \mathbf{P}(X_1) \cdot \mathbf{P}(X_2) \cdot \dots \cdot \mathbf{P}(X_d).$$

Alle Einträge der Verteilung lassen sich damit aus den d Werten $P(X_1), \dots, P(X_d)$ berechnen. Interessante Anwendungen lassen sich aber meist nicht modellieren, denn bedingte Wahrscheinlichkeiten werden trivial.¹³ Wegen

$$P(A | B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A)$$

¹³ Bei der Naive-Bayes-Methode wird die Unabhängigkeit aller Attribute angenommen und mit Erfolg auf die Textklassifikation angewendet (siehe Abschn. 8.7).

reduzieren sich alle bedingten Wahrscheinlichkeiten auf die A-priori-Wahrscheinlichkeiten. Interessanter wird die Welt, wenn nur ein Teil der Variablen unabhängig beziehungsweise unter bestimmten Bedingungen unabhängig sind. Für das Schließen in der KI sind nämlich gerade die Abhängigkeiten zwischen Variablen wichtig und müssen genutzt werden.

An einem einfachen und sehr anschaulichen Beispiel von J. Pearl [Pea88], das durch [RN03] noch bekannter wurde und mittlerweile zum KI-Grundwissen gehört, wollen wir das Schließen mit Bayes-Netzen erläutern.

Beispiel 7.7 (Alarm-Beispiel)

Der alleinstehende Bob hat zum Schutz vor Einbrechern in seinem Einfamilienhaus eine Alarmanlage installiert. Da Bob berufstätig ist, kann er den Alarm nicht hören, wenn er im Büro ist. Daher hat er die beiden Nachbarn John im linken Nachbarhaus und Mary im rechten Nachbarhaus gebeten, ihn im Büro anzurufen, wenn sie seinen Alarm hören. Nach einigen Jahren kennt Bob die Zuverlässigkeit von John und Mary gut und modelliert deren Anrufverhalten mittels bedingter Wahrscheinlichkeiten wie folgt.¹⁴

$$\begin{aligned} P(J|Al) &= 0,90 & P(M|Al) &= 0,70 \\ P(J|\neg Al) &= 0,05 & P(M|\neg Al) &= 0,01 \end{aligned}$$

Da Mary schwerhörig ist, überhört sie den Alarm öfter als John. John dagegen verwechselt manchmal den Alarm von Bob's Haus mit dem Alarm anderer Häuser. Der Alarm wird ausgelöst durch einen Einbruch. Aber auch ein (schwaches) Erdbeben kann den Alarm auslösen, was dann zu einem Fehlalarm führt, denn Bob will im Büro nur über Einbrüche informiert werden. Diese Zusammenhänge werden modelliert durch

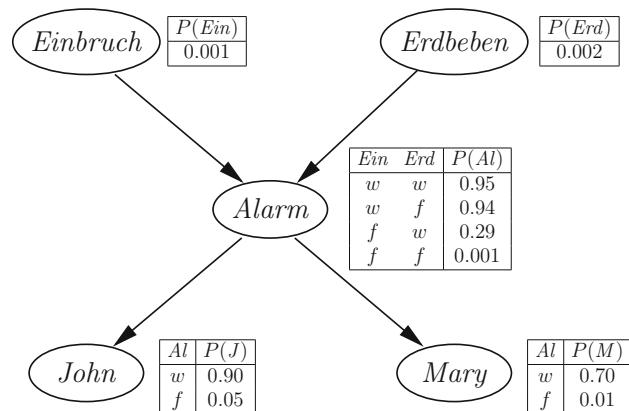
$$\begin{aligned} P(Al|Ein, Erd) &= 0,95 \\ P(Al|Ein, \neg Erd) &= 0,94 \\ P(Al|\neg Ein, Erd) &= 0,29 \\ P(Al|\neg Ein, \neg Erd) &= 0,001, \end{aligned}$$

sowie durch die A-priori-Wahrscheinlichkeiten $P(Ein) = 0,001$ und $P(Erd) = 0,002$. Diese beiden Variablen sind unabhängig, denn Erdbeben richten sich nicht nach den Gewohnheiten von Einbrechern und umgekehrt gibt es keine Vorhersagen für Erdbeben, so dass Einbrecher keine Möglichkeit haben, ihren Terminplan danach auszurichten.

An diese Wissensbasis werden nun Anfragen gestellt. Zum Beispiel könnte sich Bob für $P(Ein|J \vee M)$ oder für $P(J|Ein)$ bzw. $P(M|Ein)$ interessieren. Das heißt er will wissen, wie sensitiv die Variablen J und M auf eine Einbruchmeldung reagieren.

¹⁴ Die binären Variablen J und M stehen für die beiden Ereignisse „John ruft an“, beziehungsweise „Mary ruft an“, Al für „Alarmsirene ertönt“, Ein für „Einbruch“ und Erd für „Erdbeben“.

Abb. 7.11 Bayes-Netz für das Alarm-Beispiel mit den zugehörigen CPTs



7.4.2 Graphische Darstellung des Wissens als Bayes-Netz

Eine starke Vereinfachung der praktischen Arbeit erreicht man durch die graphische Darstellung des in Form von bedingten Wahrscheinlichkeiten formulierten Wissens. Abbildung 7.11 zeigt das zum Alarm-Beispiel passende Bayes-Netz. Jeder Knoten in dem Netz repräsentiert eine Variable und jede gerichtete Kante einen Satz von bedingten Wahrscheinlichkeiten. Die Kante von Al nach J zum Beispiel repräsentiert die beiden Werte $P(J|Al)$ und $P(J|\neg Al)$, welche in Form einer Tabelle, der so genannten CPT (engl. conditional probability table) angegeben ist. Die CPT eines Knotens enthält die bedingten Wahrscheinlichkeiten der Knotenvariable gegeben alle Knoten (Variablen), die über eingehende Kanten verbunden sind.

Beim Studium des Netzes kann man sich fragen, warum außer den vier eingezeichneten Kanten keine weiteren eingetragen sind. Bei den beiden Knoten Ein und Erd ist die Unabhängigkeit der Grund für die fehlende Kante. Da alle anderen Knoten einen Vorgängerknoten besitzen, ist hier die Antwort nicht ganz so einfach. Wir benötigen zuerst den Begriff der bedingten Unabhängigkeit.

7.4.3 Bedingte Unabhängigkeit

In Analogie zur Unabhängigkeit von Zufallsvariablen definieren wir:

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 Kombinationen der Werte für alle drei Variablen (das heißt für die Verteilung), wie man an der Schreibweise erkennt. Betrachten wir nun im Alarm-Beispiel die Knoten J und M , welche den gemeinsamen Vorgängerknoten Al besitzen. Wenn John und Mary unabhängig von einander auf einen Alarm reagieren, dann sind die beiden Variablen J und M unabhängig, gegeben Al , das heißt, es gilt

$$P(J, M | Al) = P(J | Al) \cdot P(M | Al).$$

Ist der Wert von Al bekannt, zum Beispiel weil ein Alarm ausgelöst wurde, so sind die Variablen J und M unabhängig (unter der Bedingung $Al = w$). Wegen der bedingten Unabhängigkeit der beiden Variablen J und M wird im Netz zwischen diesen beiden keine Kante eingetragen. J und M sind aber nicht unabhängig (siehe Aufgabe 7.11).

Ähnlich liegen die Verhältnisse für die beiden Variablen J und Ein , denn John reagiert nicht auf einen Einbruch sondern nur auf den Alarm. Dies ist zum Beispiel dann gegeben, wenn John wegen einer hohen Mauer Bob's Grundstück nicht einsehen, den Alarm aber hören kann. Also sind J und Ein unabhängig, gegeben Al und es gilt

$$P(J, Ein | Al) = P(J | Al) \cdot P(Ein | Al).$$

Gegeben Alarm sind außerdem noch unabhängig die Variablen J und Erd , M und Ein , sowie M und Erd . Für das Rechnen mit bedingten Unabhängigen sind die folgenden, zu obiger Definition äquivalenten Charakterisierungen hilfreich:

Satz 7.5

Folgende Gleichungen sind paarweise äquivalent, das heißt jede einzelne dieser Gleichungen beschreibt die bedingte Unabhängigkeit der Variablen A und B gegeben C :

$$P(A, B | C) = P(A | C) \cdot P(B | C) \quad (7.16)$$

$$P(A | B, C) = P(A | C) \quad (7.17)$$

$$P(B | A, C) = P(B | C) \quad (7.18)$$

Beweis Einerseits können wir unter Verwendung der bedingten Unabhängigkeit (7.16) folgern, dass

$$P(A, B, C) = P(A, B | C)P(C) = P(A | C)P(B | C)P(C)$$

gilt. Andererseits liefert die Produktregel

$$P(A, B, C) = P(A | B, C)P(B | C)P(C).$$

Also ist $P(A | B, C) = P(A | C)$ äquivalent zu (7.16). Gleichung (7.18) erhält man analog durch Vertauschung von A und B in dieser Herleitung. \square

7.4.4 Praktische Anwendung

Nun wenden wir uns wieder dem Alarm-Beispiel zu und zeigen, wie das Bayes-Netz in Abb. 7.11 zum Schließen verwendet werden kann. Bob interessiert sich zum Beispiel für die Sensitivität seiner beiden Alarmmelder John und Mary, das heißt für $P(J|Ein)$ und $P(M|Ein)$. Noch wichtiger aber sind für ihn die Werte $P(Ein|J)$ und $P(Ein|M)$ sowie $P(Ein|J, M)$. Wir starten mit $P(J|Ein)$ und berechnen

$$P(J|Ein) = \frac{P(J, Ein)}{P(Ein)} = \frac{P(J, Ein, Al) + P(J, Ein, \neg Al)}{P(Ein)} \quad (7.19)$$

und

$$P(J, Ein, Al) = P(J|Ein, Al)P(Al|Ein)P(Ein) = P(J|Al)P(Al|Ein)P(Ein), \quad (7.20)$$

wobei wir für die beiden letzten Gleichungen die Produktregel und die bedingte Unabhängigkeit von J und Ein , gegeben Al , verwendet haben. Eingesetzt in (7.19) erhalten wir

$$\begin{aligned} P(J|Ein) &= \frac{P(J|Al)P(Al|Ein)P(Ein) + P(J|\neg Al)P(\neg Al|Ein)P(Ein)}{P(Ein)} \\ &= P(J|Al)P(Al|Ein) + P(J|\neg Al)P(\neg Al|Ein). \end{aligned} \quad (7.21)$$

Hier fehlen $P(Al|Ein)$ und $P(\neg Al|Ein)$. Wir berechnen also

$$\begin{aligned} P(Al|Ein) &= \frac{P(Al, Ein)}{P(Ein)} = \frac{P(Al, Ein, Erd) + P(Al, Ein, \neg Erd)}{P(Ein)} \\ &= \frac{P(Al|Ein, Erd)P(Ein)P(Erd) + P(Al|Ein, \neg Erd)P(Ein)P(\neg Erd)}{P(Ein)} \\ &= P(Al|Ein, Erd)P(Erd) + P(Al|Ein, \neg Erd)P(\neg Erd) \\ &= 0,95 \cdot 0,002 + 0,94 \cdot 0,998 = 0,94 \end{aligned}$$

sowie $P(\neg Al|Ein) = 0,06$ und setzen in (7.21) ein, was zum Ergebnis

$$P(J|Ein) = 0,9 \cdot 0,94 + 0,05 \cdot 0,06 = 0,849$$

führt. Analog berechnet man $P(M|Ein) = 0,659$. Wir wissen nun also, dass John bei etwa 85 % aller Einbrüche anruft und Mary bei etwa 66 % aller Einbrüche. Die Wahrscheinlichkeit, dass beide anrufen, ergibt sich aufgrund der bedingten Unabhängigkeit zu

$$P(J, M|Ein) = P(J|Ein)P(M|Ein) = 0,849 \cdot 0,659 = 0,559.$$

Interessanter ist aber die Wahrscheinlichkeit für einen Anruf von John oder Mary

$$\begin{aligned} P(J \vee M|Ein) &= P(\neg(\neg J, \neg M)|Ein) = 1 - P(\neg J, \neg M|Ein) \\ &= 1 - P(\neg J|Ein)P(\neg M|Ein) = 1 - 0,051 = 0,948. \end{aligned}$$

Bob bekommt also etwa 95 % aller Einbrüche gemeldet. Um nun $P(Ein|J)$ zu berechnen, wenden wir die Bayes-Formel an, die uns

$$P(Ein|J) = \frac{P(J|Ein)P(Ein)}{P(J)} = \frac{0,849 \cdot 0,001}{0,052} = 0,016$$

liefert. Offenbar haben nur etwa 1,6 % aller Anrufe von John einen Einbruch als Ursache. Da die Wahrscheinlichkeit für Fehlalarme bei Mary fünfmal geringer ist als bei John, erhalten wir mit $P(Ein|M) = 0,056$ eine wesentlich höhere Sicherheit bei einem Anruf von Mary. Wirkliche Sorgen um sein Eigenheim sollte sich Bob aber erst dann machen, wenn beide anrufen, denn $P(Ein|J, M) = 0,284$ (siehe Aufgabe 7.11).

In (7.21) haben wir mit

$$P(J|Ein) = P(J|Al)P(Al|Ein) + P(J|\neg Al)P(\neg Al|Ein)$$

gezeigt, wie man eine neue Variable „einschieben“ kann. Dieser Zusammenhang gilt allgemein für zwei Variablen A und B bei Einführung einer weiteren Variablen C und wird **Konditionierung** genannt:

$$P(A|B) = \sum_c P(A|B, C=c)P(C=c|B).$$

Wenn nun außerdem noch A und B bedingt unabhängig gegeben C sind, so vereinfacht sich die Formel zu

$$P(A|B) = \sum_c P(A|C=c)P(C=c|B).$$

7.4.5 Software für Bayes-Netze

Anhand des Alarm-Beispiels stellen wir zwei Werkzeuge kurz vor. Das System PIT ist schon bekannt. Wir geben unter <http://www.pit-systems.de> die Werte aus den CPTs in PIT-Syntax in das Online-Eingabefenster ein und starten PIT. Nach der in Abb. 7.12 dargestellten Eingabe erhalten wir als Antwort:

```
P([Einbruch=t] | [John=t] AND [Mary=t]) = 0.2841.
```

Obwohl PIT kein klassisches Bayes-Netz-Werkzeug ist, können beliebige bedingte Wahrscheinlichkeiten und Anfragen eingegeben werden und PIT berechnet richtige Ergebnisse. Man kann nämlich zeigen [Sch96], dass bei der Angabe der CPTs oder äquivalenter Regeln aus dem MaxEnt-Prinzip die gleichen bedingten Unabhängigkeiten wie bei einem Bayes-Netz folgen und somit auch die gleichen Antworten berechnet werden.

```

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] AND [Erdbeben=t]) = 0.95;
6 P([Alarm=t] | [Einbruch=t] AND [Erdbeben=f]) = 0.94;
7 P([Alarm=t] | [Einbruch=f] AND [Erdbeben=t]) = 0.29;
8 P([Alarm=t] | [Einbruch=f] AND [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 Alarm-Beispiel

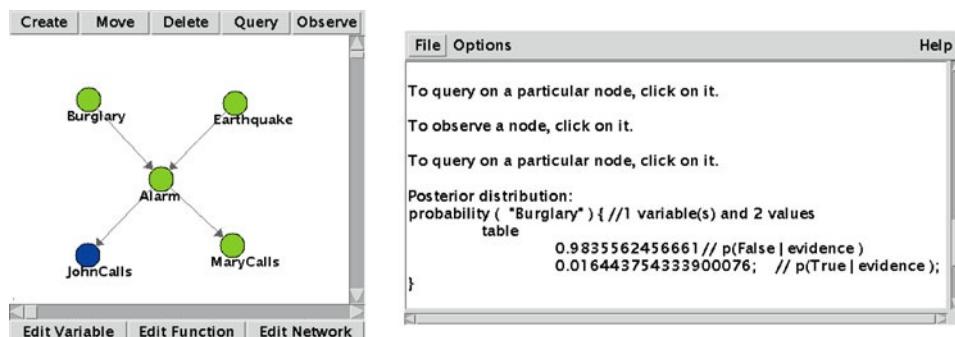


Abb. 7.13 Die Benutzeroberfläche von JavaBayes: Links der graphische Editor und rechts die Konsole, auf der die Antworten ausgegeben werden

Als nächstes betrachten wir JavaBayes, ein klassisches, auch im Internet verfügbares System mit der in Abb. 7.13 dargestellten graphischen Oberfläche. Mit dem graphischen Netzwerkeditor können Knoten und Kanten bearbeitet und auch die Werte in den CPTs editiert werden. Außerdem können mit „Observe“ die Werte von Variablen festgelegt und mit „Query“ die Werte von anderen Variablen abgefragt werden. Die Antworten auf Anfragen erscheinen dann im Konsolenfenster. JavaBayes ist frei verfügbar, auch als JavaApplet [Coz98].

Wesentlich mächtiger und komfortabler ist das professionell vertriebene System Hugin. Zum Beispiel kann Hugin neben diskreten auch stetige Variablen verwenden. Es kann auch Bayes-Netze lernen, das heißt das Netz vollautomatisch aus statistischen Daten generieren (siehe Abschn. 8.6).

7.4.6 Entwicklung von Bayes-Netzen

Ein kompaktes Bayes-Netz ist sehr übersichtlich und für den Betrachter wesentlich informativer als eine vollständige Wahrscheinlichkeitsverteilung. Außerdem benötigt es viel weniger Speicherplatz. Bei den Variablen v_1, \dots, v_n mit jeweils $|v_1|, \dots, |v_n|$ verschiedenen Werten hat die Verteilung insgesamt

$$\prod_{i=1}^n |v_i| - 1$$

unabhängige Einträge. Im Alarm-Beispiel sind die Variablen alle binär, also ist für alle Variablen $|v_i| = 2$. Die Verteilung hat also $2^5 - 1 = 31$ unabhängige Einträge. Um für das Bayes-Netz die Zahl der unabhängigen Einträge zu berechnen, muss die Gesamtzahl aller Einträge aller CPTs bestimmt werden. Für einen Knoten v_i mit den k_i Elternknoten e_{i1}, \dots, e_{ik_i} besitzt die zugehörige CPT

$$(|v_i| - 1) \prod_{j=1}^{k_i} |e_{ij}|$$

Einträge. Alle CPTs des Netzes zusammen haben dann

$$\sum_{i=1}^n (|v_i| - 1) \prod_{j=1}^{k_i} |e_{ij}| \quad (7.22)$$

Einträge.¹⁵ Für das Alarm-Beispiel ergeben sich damit

$$2 + 2 + 4 + 1 + 1 = 10$$

unabhängige Einträge, welche das Netz eindeutig beschreiben. Der Vergleich der Speicherkomplexität zwischen der vollständigen Verteilung und dem Bayes-Netz wird anschaulicher, wenn man annimmt, alle n Variablen haben die gleiche Zahl b an Werten und jeder Knoten hat k Elternknoten. Dann vereinfacht sich (7.22) und alle CPTs zusammen besitzen $n(b - 1)b^k$ Einträge. Die vollständige Verteilung enthält $b^n - 1$ Einträge. Einen deutlichen Gewinn erzielt man also dann, wenn die mittlere Zahl der Elternknoten viel kleiner als die Zahl der Variablen ist. Das bedeutet, dass die Knoten nur lokal vernetzt sind. Durch die lokale Vernetzung wird auch eine Modularisierung des Netzes erreicht, die – ähnlich wie bei der Softwareentwicklung – mit einer Komplexitätsreduktion einhergeht. Im Alarm-Beispiel etwa trennt der Alarm-Knoten die Knoten *Ein* und *Erd* von den Knoten *J* und *M*. Ganz schön erkennt man dies auch am Beispiel von LEXMED.

¹⁵ Für den Fall eines Knotens ohne Vorgänger ist das Produkt in dieser Summe leer. Hierfür setzen wir den Wert 1 ein, denn die CPT für Knoten ohne Vorgänger enthält mit der A-priori-Wahrscheinlichkeit genau einen Wert.

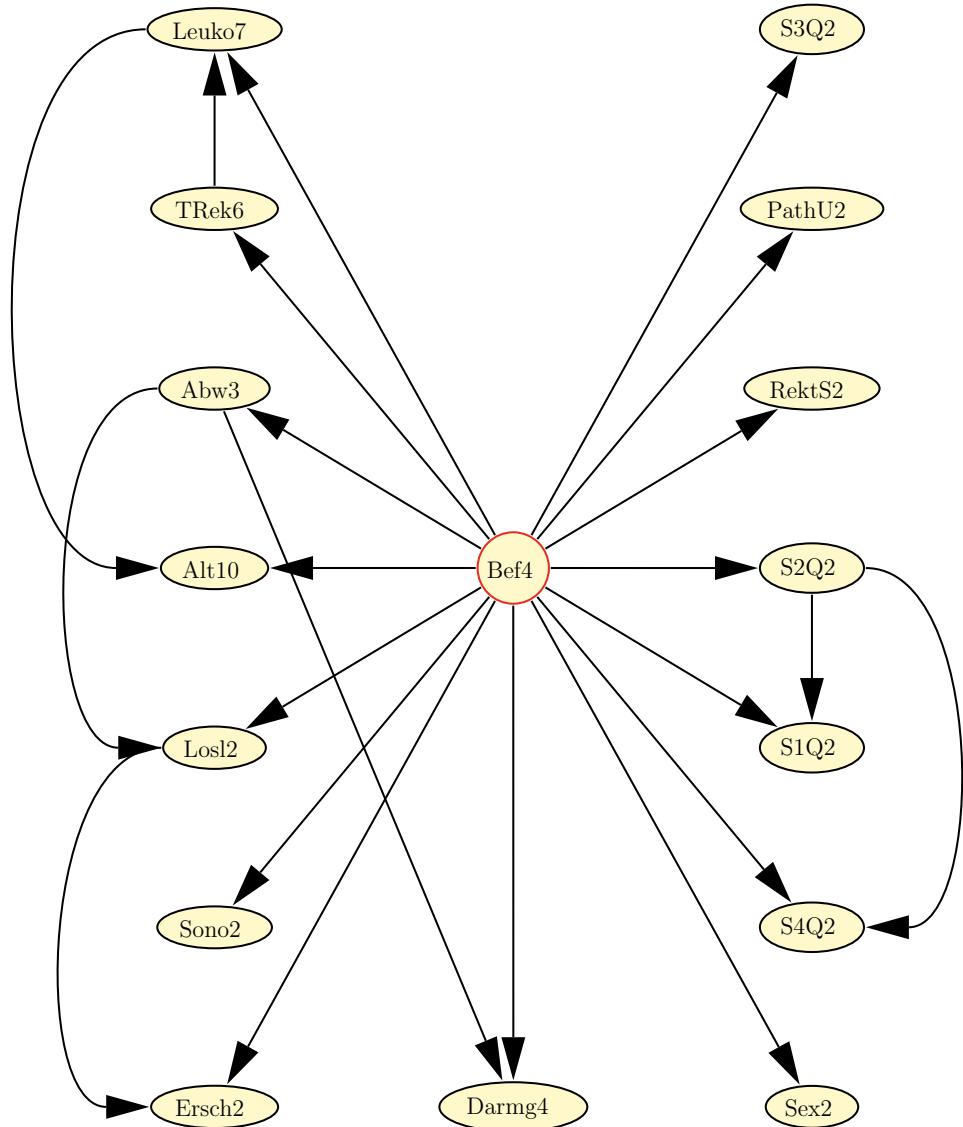


Abb. 7.14 Bayes-Netz für die LEXMED-Anwendung

LEXMED als Bayes-Netz

Das in Abschn. 7.3 beschriebene System LEXMED kann auch als Bayes-Netz modelliert werden. Der Unabhängigkeitsgraph in Abb. 7.8 lässt sich durch Ausrichtung der äußereren, schwach eingezeichneten Kanten als Bayes-Netz interpretieren. In Abb. 7.14 ist das resultierende Netz dargestellt.

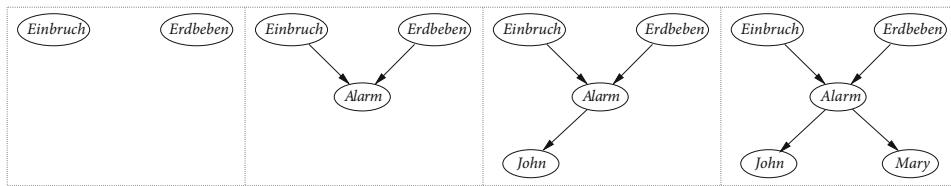


Abb. 7.15 Schrittweiser Aufbau des Alarm-Netzes unter Beachtung der Kausalität

In Abschn. 7.3.2 wurde als Mächtigkeit der Verteilung für LEXMED der Wert 20.643.839 berechnet. Das Bayes-Netz lässt sich hingegen mit nur 521 Werten vollständig beschreiben. Dieser Wert ergibt sich, wenn man die Variablen aus Abb. 7.14 von links oben im Gegen-uhzeigersinn in (7.22) einträgt. In der Reihenfolge (Leuko, TRek, Abw, Alt, Losl, Sono, Ersch, Darmg, Sex, S4Q, S1Q, S2Q, RektS, PathU, S3Q, Bef4) berechnet man

$$\sum_{i=1}^n (|v_i| - 1) \prod_{j=1}^{k_i} |e_{ij}| = 6 \cdot 6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 7 \cdot 4 + 1 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 2 \cdot 4 + 3 \cdot 3 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 \cdot 2 + 1 \cdot 4 + 1 \cdot 4 + 1 \cdot 4 + 1 = 521.$$

An diesem Beispiel erkennt man sehr schön, dass es für reale Anwendungen praktisch unmöglich ist, eine vollständige Verteilung aufzubauen. Ein Bayes-Netz mit 22 Knoten und 521 Wahrscheinlichkeitswerten hingegen ist noch handlich.

Kausalität und Netzstruktur

Beim Aufbau eines Bayes-Netzes wird normalerweise zweistufig vorgegangen.

1. Entwurf der Netzwerkstruktur Dieser Schritt erfolgt meist manuell und wird im Folgenden beschrieben.

2. Eintragen der Wahrscheinlichkeiten in die CPTs Manuelles Ermitteln der Werte wird im Fall von vielen Variablen sehr mühsam. Falls (wie zum Beispiel bei LEXMED) eine Datenbank verfügbar ist, kann dieser Schritt automatisiert werden, indem man durch Zählen von Häufigkeiten die CPT-Einträge schätzt.

An dem Alarm-Beispiel beschreiben wir nun den Aufbau des Netzes (siehe Abb. 7.15). Zu Beginn stehen die beiden Ursachen *Einbruch* und *Erdbeben* und die beiden Symptome *John* und *Mary* fest. Da *John* und *Mary* aber nicht direkt auf *Einbruch* oder *Erdbeben* reagieren, sondern nur auf den *Alarm*, bietet es sich an, diesen als weitere, für Bob nicht beobachtbare, Variable hinzuzunehmen. Zur Bestimmung der erforderlichen Kanten im Netz starten wir nun mit den Ursachen, das heißt, mit den Variablen, die keine Elternknoten besitzen. Zuerst wählen wir *Einbruch* und als nächstes *Erdbeben*. Nun muss geprüft werden, ob *Erdbeben* von *Einbruch* unabhängig ist. Dies ist gegeben, also wird keine Kante von *Einbruch*

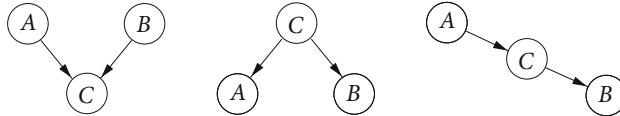


Abb. 7.16 Zwischen zwei Knoten A und B wird keine Kante eingetragen, wenn sie unabhängig (*links*) oder bedingt unabhängig sind (*Mitte, rechts*)

nach *Erdbeben* eingetragen. Weil *Alarm* direkt von *Einbruch* und *Erdbeben* abhängt, wird diese Variable als nächste gewählt und je eine Kante von *Einbruch* und *Erdbeben* nach *Alarm* eingetragen. Nun wählen wir *John*. Da *Alarm* und *John* nicht unabhängig sind, wird eine Kante von *Alarm* nach *John* eingetragen. Gleiches gilt für *Mary*. Nun muss noch geprüft werden, ob *John* bedingt unabhängig ist von *Einbruch* gegeben *Alarm*. Ist dies nicht der Fall, so muss noch eine Kante von *Einbruch* zu *John* eingefügt werden. Genauso muss noch geprüft werden, ob Kanten von *Erdbeben* zu *John* und von *Einbruch* oder *Erdbeben* zu *Mary* benötigt werden. Aufgrund der bedingten Unabhängigkeit sind diese vier Kanten nicht nötig. Auch wird keine Kante zwischen *John* und *Mary* benötigt, denn *John* und *Mary* sind bedingt unabhängig gegeben *Alarm*.

Die Struktur des Bayes-Netzes hängt stark von der gewählten Variablenreihenfolge ab. Wird die Reihenfolge der Variablen im Sinne der Kausalität, angefangen von den Ursachen, hin zu den Diagnosevariablen, gewählt, dann erhält man ein einfaches Netz. Andernfalls kann das Netz wesentlich mehr Kanten enthalten. Solche nicht kausalen Netze sind oft sehr schwer verständlich und die Komplexität des Schließens wird erhöht. Der Leser möge dies anhand von Aufgabe 7.11 nachvollziehen.

7.4.7 Semantik von Bayes-Netzen

Wie wir im vorhergehenden Abschnitt gesehen haben, wird in einem Bayes-Netz zwischen zwei Variablen A und B keine Kante eingetragen, wenn diese unabhängig sind oder bedingt unabhängig, gegeben eine dritte Variable C . Diese Situation ist dargestellt in Abb. 7.16.

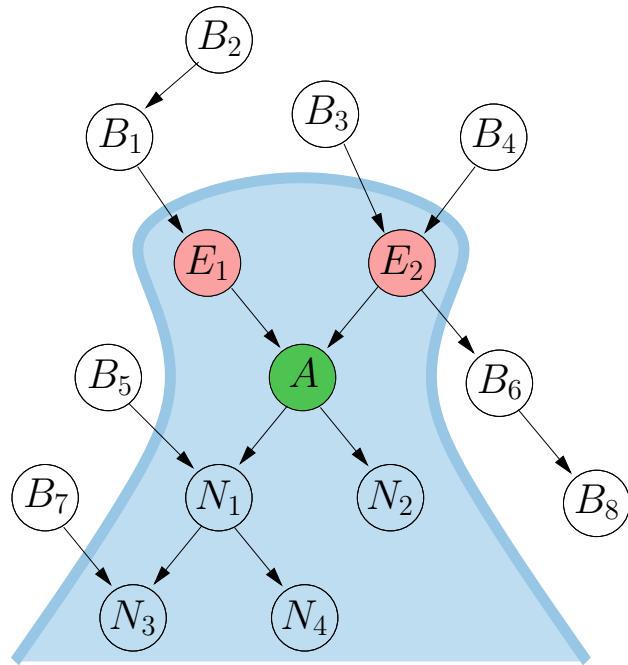
Wir fordern nun, dass das Bayes-Netz keine Zyklen hat, und nehmen an, die Variablen sind so nummeriert, dass keine Variable einen Nachfolger mit kleinerer Nummer hat. Wenn das Netz keine Zyklen hat, ist dies immer möglich.¹⁶ Unter Verwendung aller bedingten Unabhängigkeit gilt dann

$$P(X_n | X_1, \dots, X_{n-1}) = P(X_n | \text{Eltern}(X_n)).$$

Die Aussage dieser Gleichung ist also die, dass eine beliebige Variable X_i in einem Bayes-Netz bedingt unabhängig von ihren Vorfahren, gegeben ihre Eltern, ist. Es gilt sogar die in Abb. 7.17 dargestellte, etwas allgemeinere Aussage

¹⁶ Wenn zum Beispiel die drei Knoten X_1, X_2, X_3 einen Zyklus bilden, dann gibt es die Kanten $(X_1, X_2), (X_2, X_3)$ und (X_3, X_1) , wobei X_3 den Nachfolger X_1 hat.

Abb. 7.17 Beispiel für die bedingte Unabhängigkeit in einem Bayes-Netz. Sind die Elternknoten E_1 und E_2 gegeben, so sind alle Nichtnachfolger B_1, \dots, B_8 unabhängig von A



Satz 7.6

Ein Knoten in einem Bayes-Netz ist bedingt unabhängig von allen Nicht-Nachfolgern, gegeben seine Eltern.

Nun sind wir in der Lage, die Kettenregel (7.1) stark zu vereinfachen:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) = \prod_{i=1}^n P(X_i | \text{Eltern}(X_i)).$$

Mit Hilfe dieser Regel hätten wir zum Beispiel (7.20)

$$P(J, Ein, Al) = P(J | Al)P(Al | Ein)P(Ein)$$

direkt hinschreiben können.

Die wichtigsten Begriffe und Grundlagen von Bayes-Netzen sind nun bekannt und wir können diese zusammenfassen [Jen01]:

Definition 7.7

Ein Bayes-Netz ist definiert durch:

- Eine Menge von Variablen und einer Menge von gerichteten Kanten zwischen diesen Variablen.
- Jede Variable hat endlich viele mögliche Werte.
- Die Variablen zusammen mit den Kanten stellen einen gerichteten azyklischen Graphen (engl. directed acyclic graph, DAG) dar. Ein DAG ist ein gerichteter Graph ohne Zyklen, das heißt ohne Pfade der Form (A, \dots, A) .
- Zu jeder Variablen A ist die CPT, das heißt die Tabelle der bedingten Wahrscheinlichkeiten $P(A | \text{Eltern}(A))$, angegeben.

Zwei Variablen A und B heißen **bedingt unabhängig** gegeben C , wenn $P(A, B | C) = P(A | C) \cdot P(B | C)$, bzw. wenn $P(A | B, C) = P(A | C)$.

Neben den grundlegenden Rechenregeln für Wahrscheinlichkeiten gelten folgende Regeln:

Bayes-Formel:

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}.$$

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:

$$P(A | B) = \sum_c P(A | B, C = c) P(C = c | B).$$

Eine Variable in einem Bayes-Netz ist bedingt unabhängig von allen Nicht-Nachfolge-Variablen, gegeben ihre Eltern-Variablen. Wenn X_1, \dots, X_{n-1} keine Nachfolger von X_n sind, gilt $P(X_n | X_1, \dots, X_{n-1}) = P(X_n | \text{Eltern}(X_n))$. Diese Bedingung muss beim Aufbau eines Netzes beachtet werden.

Beim Aufbau eines Bayes-Netzes sollten die Variablen in Sinne der Kausalität angeordnet werden. Zuerst die Ursachen, dann die verdeckten 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 der d-Separation für Bayes-Netze eingeführt, woraus dann ein zu Satz 7.6 analoger Satz folgt. Wir verzichten auf die Einführung dieses Begriffes und erreichen dadurch eine etwas einfachere, aber theoretisch nicht ganz so saubere Darstellung.

7.5 Zusammenfassung

Entsprechend dem schon lange anhaltenden Trend hin zu probabilistischen Systemen in der KI haben wir die Wahrscheinlichkeitslogik zum Schließen mit unsicherem Wissen eingeführt. Nach der Einführung der Sprache – Aussagenlogik erweitert um Wahrscheinlichkeiten beziehungsweise Wahrscheinlichkeitsintervalle – wählten wir als Einstieg den ganz natürlichen, wenn auch noch nicht üblichen Zugang über die Methode der maximalen Entropie und zeigten dann unter anderem, wie man mit dieser Methode auch das nichtmonotone Schließen modellieren kann. Die Bayes-Netze wurden danach, sozusagen als ein Spezialfall der MaxEnt-Methode, eingeführt.

Warum sind die Bayes-Netze ein Spezialfall von MaxEnt? Beim Aufbau eines Bayes-Netzes werden Unabhängigkeitsannahmen gemacht, die für die MaxEnt-Methode nicht notwendig sind. Außerdem müssen beim Aufbau eines Bayes-Netzes alle CPTs ganz gefüllt werden, damit eine vollständige Wahrscheinlichkeitsverteilung aufgebaut wird. Sonst ist das Schließen nicht oder nur eingeschränkt möglich. Bei MaxEnt hingegen kann der Entwickler alles Wissen, das er zur Verfügung hat, in Form von Wahrscheinlichkeiten formulieren. MaxEnt vervollständigt dann das Modell und erzeugt die Verteilung. Sogar wenn, zum Beispiel bei der Befragung eines Experten, nur sehr vage Aussagen verfügbar sind, können diese mit MaxEnt angemessen modelliert werden. Eine Aussage etwa wie „Ich bin mir ziemlich sicher, dass A gilt.“ kann dann zum Beispiel mittels $P(A) \in [0,6,1]$ als Wahrscheinlichkeitsintervall modelliert werden. Beim Aufbau eines Bayes-Netzes muss ein fester Wert für die Wahrscheinlichkeit angegeben werden, unter Umständen sogar durch Raten. Das heißt aber, der Experte oder der Entwickler stecken ad hoc Information in das System. Ein weiterer Vorteil von MaxEnt ist die Möglichkeit, (fast) beliebige Aussagen zu formulieren. Beim Bayes-Netz müssen die CPTs gefüllt werden.

Die Freiheit, die der Entwickler bei der Modellierung mit MaxEnt hat, kann, insbesondere für den Anfänger, auch ein Nachteil sein, denn im Gegensatz zur Bayesschen Vorgehensweise fehlen klare Vorgaben, welches Wissen modelliert werden soll. Bei der Entwicklung eines Bayes-Netzes ist die Vorgehensweise ganz klar: Im Sinne der kausalen Abhängigkeiten, von den Ursachen hin zu den Wirkungen, wird mittels Prüfung bedingter Unabhängigkeit¹⁷ eine Kante nach der anderen in das Netz eingetragen. Am Ende werden dann alle CPTs mit Werten gefüllt.

Hier bietet sich aber folgende interessante Kombination der beiden Methoden an: Man beginnt im Sinne der Bayesschen Methodik, ein Netz aufzubauen, trägt entsprechend alle

¹⁷ Die auch nicht immer ganz einfach sein kann.

Kanten ein und füllt dann die CPTs mit Werten. Sollten bestimmte Werte für die CPTs nicht verfügbar sein, so können sie durch Intervalle ersetzt werden oder durch andere Formeln der Wahrscheinlichkeitslogik. Natürlich hat solch ein Netz – besser: eine Regelmenge – nicht mehr die spezielle Semantik eines Bayes-Netzes. Es muss dann von einem MaxEnt-System vervollständigt und abgearbeitet werden.

Die Möglichkeit, bei Verwendung von MaxEnt beliebige Regelmengen vorzugeben, hat aber auch eine Kehrseite. Ähnlich wie in der Logik können solche Regelmengen inkonsistent sein. Zum Beispiel sind die beiden Regeln $P(A) = 0,7$ und $P(A) = 0,8$ inkonsistent. Das MaxEnt-System PIT zum Beispiel erkennt zwar die Inkonsistenz, kann aber keine Hinweise zur Beseitigung machen.

Als eine klassische Anwendung für das Schließen mit unsicherem Wissen haben wir das medizinische Expertensystem LEXMED vorgestellt und gezeigt, wie dieses mittels MaxEnt und Bayes-Netzen modelliert und implementiert werden kann und wie man mit diesen Werkzeugen die in der Medizin etablierten, aber zu schwachen linearen Score-Systeme ablösen kann.¹⁸

Am Beispiel von LEXMED haben wir gezeigt, dass es möglich ist, ein Expertensystem zum Schließen mit Unsicherheit zu bauen, das basierend auf den Daten einer Datenbank Wissen lernen kann. Einen Einblick in die Methoden des Lernens von Bayes-Netzen werden wir in Kap. 8 geben, nachdem die dazu benötigten Grundlagen des maschinellen Lernens bereit stehen.

Das Bayessche Schließen ist heute ein eigenständiges großes Gebiet, das wir hier nur kurz anreißen konnten. Ganz weggelassen haben wir den Umgang mit stetigen Variablen. Für den Fall von normalverteilten Zufallsvariablen gibt es hier Verfahren und Systeme. Bei beliebigen Verteilungen ist hingegen die Berechnungskomplexität ein großes Problem. Auch gibt es neben den stark auf der Kausalität basierenden gerichteten Netzen ungerichtete Netze. Damit verbunden ist eine Diskussion über den Sinn von Kausalität in Bayes-Netzen. Der interessierte Leser wird verwiesen auf die ausgezeichneten Lehrbücher, unter anderem [Pea88, Jen01, Whi96, DHS01] sowie auf die Konferenzbände der jährlichen Konferenz der *Association for Uncertainty in Artificial Intelligence (AUAI)* (<http://www.auai.org>).

7.6 Übungen

Aufgabe 7.1 Beweisen Sie die Aussagen von Satz 7.1.

Aufgabe 7.2 Der Hobbygärtner Max will seine Jahresernte von Erbsen statistisch analysieren. Er misst für jede gepflückte Erbsenschote Länge x_i in cm und Gewicht y_i in Gramm.

¹⁸ In Abschn. 8.7, beziehungsweise in Aufgabe 8.17 werden wir zeigen, dass die Scores äquivalent sind zum Spezialfall Naive-Bayes, das heißt zur Annahme, alle Symptome sind bedingt unabhängig gegeben die Diagnose.

Er teilt die Erbsen in 2 Klassen, die guten und die tauben (leere Schoten). Die Messdaten (x_i, y_i) sind:

gute Erbsen:	$\begin{array}{cccccccccc} x & 1 & 2 & 2 & 3 & 3 & 4 & 4 & 5 & 6 \\ y & 2 & 3 & 4 & 4 & 5 & 5 & 6 & 6 & 6 \end{array}$
--------------	--

taube Erbsen:	$\begin{array}{cccccc} x & 4 & 6 & 6 & 7 \\ y & 2 & 2 & 3 & 3 \end{array}$
---------------	--

- a) Berechnen Sie aus den Daten die Wahrscheinlichkeiten $P(y > 3 | Klasse = gut)$ und $P(y \leq 3 | Klasse = gut)$. Verwenden sie dann die Bayes-Formel zur Bestimmung von $P(Klasse = gut | y > 3)$ und $P(Klasse = gut | y \leq 3)$.
- b) Welche der in Teilaufgabe a berechneten Wahrscheinlichkeiten widerlegt die Aussage: „Alle guten Erbsen sind schwerer als 3 Gramm.“

Aufgabe 7.3 Anhand von zwei einfachen Wetterwerten vom Morgen eines Tages soll das Wetter am Nachmittag vorhergesagt werden. Die klassische Wahrscheinlichkeitsrechnung benötigt dazu ein vollständiges Modell, wie es in folgender Tabelle angegeben ist.

Him	Bar	Nied	$P(Him, Bar, Nied)$
klar	steigt	trocken	0,40
klar	steigt	regen	0,07
klar	fällt	trocken	0,08
klar	fällt	regen	0,10
bewölkt	steigt	trocken	0,09
bewölkt	steigt	regen	0,11
bewölkt	fällt	trocken	0,03

Him: Himmel ist morgens klar oder bewölkt

Bar: Barometer steigt oder fällt morgens

Nied: Regen oder trocken nachmittags

- a) Wieviele Ereignisse hat die Verteilung für diese drei Variablen?
- b) Berechnen Sie $P(Nied = trocken | Him = klar, Bar = steigt)$.
- c) Berechnen Sie $P(Nied = regen | Him = bewölkt)$.
- d) Was würden Sie tun, wenn in der Tabelle die letzte Zeile fehlen würde?

Aufgabe 7.4 * In einer Quizshow im Fernsehen muss der Kandidat eine von drei geschlossenen Türen auswählen. Hinter einer Tür wartet der Preis, ein Auto, hinter den beiden anderen stehen Ziegen. Der Kandidat wählt eine Tür, z. B. Nummer eins. Der Moderator, der weiß, wo das Auto steht, öffnet eine andere Tür, z. B. Nummer drei und es erscheint eine Ziege. Der Kandidat erhält nun nochmal die Möglichkeit, zwischen den beiden verbleibenden Türen (eins und zwei) zu wählen. Was ist aus seiner Sicht die bessere Wahl? Bei der gewählten Tür zu bleiben oder auf die andere noch geschlossene Tür zu wechseln?

Aufgabe 7.5 Zeigen Sie mit Hilfe der Methode der Lagrangemultiplikatoren, dass ohne explizite Nebenbedingungen die Gleichverteilung $p_1 = p_2 = \dots = p_n = 1/n$ das Entropiemaximum darstellt. Vergessen Sie nicht die implizit immer vorhandene Nebenbedingung $p_1 + p_2 + \dots + p_n = 1$. Wie kann man dieses Resultat auch mittels Indifferenz zeigen?

Aufgabe 7.6 Verwenden Sie das System PIT (<http://www.pit-systems.de>) oder SPIRIT (<http://www.xspirit.de>), um die MaxEnt-Lösung für $P(B)$ unter den Randbedingungen $P(A) = \alpha$ und $P(B|A) = \beta$ zu berechnen. Welchen Nachteil von PIT gegenüber dem Rechnen von Hand erkennen Sie hier?

Aufgabe 7.7 Gegeben seien die Randbedingungen $P(A) = \alpha$ und $P(A \vee B) = \beta$. Berechnen Sie manuell mit der MaxEnt-Methode $P(B)$. Verwenden Sie PIT, um Ihre Lösung zu überprüfen.

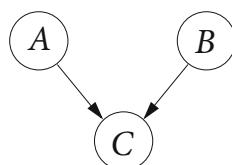
Aufgabe 7.8 * Gegeben seien die Randbedingungen aus den (7.10), (7.11) und (7.12): $p_1 + p_2 = \alpha$, $p_1 + p_3 = \gamma$, $p_1 + p_2 + p_3 + p_4 = 1$. Zeigen Sie, dass $p_1 = \alpha\gamma$, $p_2 = \alpha(1 - \gamma)$, $p_3 = \gamma(1 - \alpha)$, $p_4 = (1 - \alpha)(1 - \gamma)$ das Entropiemaximum unter diesen Nebenbedingungen darstellt.

Aufgabe 7.9 * Ein probabilistisches Verfahren berechnet für ankommende Emails deren Spam-Wahrscheinlichkeit p . Zur Klassifikation der Emails in die Klassen *Löschen* und *Lesen* wird dann auf das Ergebnis, das heißt auf den Vektor $(p, 1 - p)$, eine Kostenmatrix angewendet.

- Geben Sie eine Kostenmatrix (2×2 -Matrix) an für die Spam-Filterung. Nehmen Sie hierzu an, für das Löschen einer Spam-Mail durch den Benutzer fallen Kosten in Höhe von 10 Cent an. Für das Wiederbeschaffen gelöschter Mails, bzw. den Verlust einer Mail, fallen Kosten in Höhe von 10 Euro an (vergleiche Beispiel 1.1 beziehungsweise Aufgabe 1.7).
- Zeigen Sie, dass für den Fall einer 2×2 -Matrix die Anwendung der Kostenmatrix äquivalent ist zur Anwendung einer Schwelle auf die Spam-Wahrscheinlichkeit und bestimmen Sie die Schwelle.

Aufgabe 7.10 Gegeben sei ein Bayesnetz mit den drei binären Variablen A, B, C und $P(A) = 0,2$, $P(B) = 0,9$, sowie der angegebenen CPT.

A	B	$P(C)$
w	f	0,1
w	w	0,2
f	w	0,9
f	f	0,4



- Berechnen Sie $P(A|B)$.
- Berechnen Sie $P(C|A)$.

Aufgabe 7.11 Für das Alarm-Beispiel (Beispiel 7.7) sollen verschiedene bedingte Wahrscheinlichkeiten berechnet werden.

- Berechnen Sie die A-priori-Wahrscheinlichkeiten $P(Al)$, $P(J)$, $P(M)$.
- Berechnen Sie unter Verwendung von Produktregel, Marginalisierung, Kettenregel und bedingter Unabhängigkeit $P(M|Ein)$.
- Verwenden Sie nun die Bayes-Formel zur Berechnung von $P(Ein|M)$.
- Berechnen Sie $P(Al|J,M)$ und $P(Ein|J,M)$.
- Zeigen Sie, dass die Variablen J und M nicht unabhängig sind.
- Überprüfen Sie alle Ihre Ergebnisse mit JavaBayes und mit PIT (siehe [Ert07] → Demoprogramme).
- Entwerfen Sie ein Bayes-Netz für das Alarm-Beispiel, jedoch mit der geänderten Variablenreihenfolge M, Al, Erd, Ein, J . Entsprechend der Semantik des Bayes-Netzes sind nur die notwendigen Kanten einzutragen. (Hinweis: Die hier angegebene Variablenreihenfolge entspricht NICHT der Kausalität. Daher ist es nicht einfach, intuitiv die bedingten Unabhängigkeiten zu ermitteln.)
- Im ursprünglichen Bayes-Netz des Alarm-Beispiels wird der Knoten Erdbeben gestrichen. Welche CPTs ändern sich? (Warum genau diese?)
- Berechnen Sie die CPT des Alarm-Knotens im neuen Netz.

Aufgabe 7.12 Für eine per Dynamo angetriebene Fahrradlichtanlage soll mit Hilfe eines Bayes-Netzes ein Diagnosesystem erstellt werden. Gegeben sind die Variablen in folgender Tabelle.

Abk.	Bedeutung	Werte
<i>Li</i>	Licht brennt	w/f
<i>Str</i>	Straßenzustand	trocken, nass, schneeb.
<i>La</i>	Dynamolauftrad abgenutzt	w/f
<i>R</i>	Dynamo rutscht durch	w/f
<i>Sp</i>	Dynamo liefert Spannung	w/f
<i>B</i>	Glühbirnen o.k.	w/f
<i>K</i>	Kabel o.k.	w/f

Folgende Variablen seien paarweise unabhängig: Str, La, B, K . Außerdem sind unabhängig: $(R, B), (R, K), (Sp, B), (Sp, K)$ und es gelten folgende Gleichungen:

$$\begin{aligned} P(Li|Sp, R) &= P(Li|Sp) \\ P(Sp|R, Str) &= P(Sp|R) \\ P(Sp|R, La) &= P(Sp|R) \end{aligned}$$

Str	La	Sp	B	K	$P(Li)$
		w	w	w	0,99
		w	w	f	0,01
		w	f	w	0,01
		w	f	f	0,001
		f	w	w	0,3
		f	w	f	0,005
		f	f	w	0,005
		f	f	f	0

- a) Zeichnen Sie in den Graphen (unter Berücksichtigung der Kausalität) alle Kanten ein.
- b) Tragen Sie in den Graphen alle noch fehlenden CPTs (Tabellen bedingter Wahrscheinlichkeiten) ein. Setzen Sie plausible Werte für die Wahrscheinlichkeiten frei ein.
- c) Zeigen Sie, dass das Netz keine Kante (Str, Li) enthält.
- d) Berechnen Sie $P(Sp | Str = \text{schneeb})$.

Definiert man den Begriff der KI wie im Buch von Elaine Rich [[Ric83](#)]

Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better.

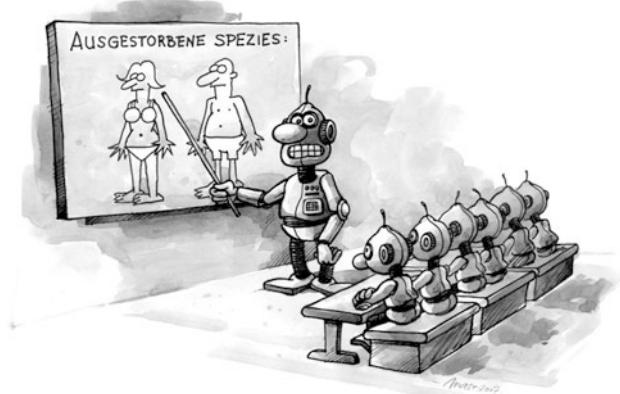
und bedenkt, dass die Computer uns Menschen insbesondere bezüglich der Lernfähigkeit weit unterlegen sind, dann folgt daraus, dass die Erforschung der Mechanismen des Lernens und die Entwicklung maschineller Lernverfahren eines der wichtigsten Teilgebiete der KI darstellt.

Die Forderung nach maschinellen Lernverfahren ergibt sich aber auch aus dem Blickwinkel des Software-Entwicklers, der zum Beispiel das Verhalten eines autonomen Roboters programmieren soll. Die Struktur des intelligenten Verhaltens kann hierbei so komplex werden, dass es auch mit modernen Hochsprachen wie Prolog oder Python¹ sehr schwierig oder sogar unmöglich wird, dieses annähernd optimal zu programmieren. Ähnlich wie wir Menschen lernen, werden auch heute schon bei der Programmierung von Robotern maschinelle Lernverfahren eingesetzt (siehe Kap. 10 bzw. [[RGH⁺06](#)]), oft auch in einer hybriden Mischung aus programmiertem und gelerntem Verhalten.

Aufgabe dieses Kapitels ist es, die wichtigsten maschinellen Lernverfahren und deren Anwendung zu beschreiben. Nach einer Einführung in das Thema in diesem Abschnitt folgen in den nächsten Abschnitten einige wichtige grundlegende Lernverfahren. Parallel dazu werden Theorie und Begriffsbildung entwickelt. Abgeschlossen wird das Kapitel mit einer Zusammenfassung und Übersicht über die verschiedenen Verfahren und deren Anwendung. Hierbei beschränken wir uns in diesem Kapitel auf Verfahren des Lernens mit und ohne Lehrer. Die neuronalen Netze als wichtige Klasse von Lernverfahren werden in Kap. 9 behandelt. Auch dem Lernen durch Verstärkung wird aufgrund seiner Sonder-

¹ Python ist eine moderne Skriptsprache mit sehr übersichtlicher Syntax, mächtigen Datentypen und umfangreicher Standardbibliothek, die sich für diesen Zweck anbietet.

Abb. 8.1 Maschinelles Lernen mit Lehrer ...



stellung und seiner wichtigen Rolle für autonome Roboter ein eigenes Kapitel (Kap. 10) gewidmet.

Was ist Lernen? Das Lernen von Vokabeln einer Fremdsprache oder Fachbegriffen oder auch das Auswendiglernen eines Gedichts fällt vielen Menschen schwer. Für Computer hingegen sind diese Aufgaben ganz einfach, denn im Wesentlichen muss der Text in eine Datei gespeichert werden. Damit ist das Auswendiglernen für die KI uninteressant. Im Gegensatz dazu erfolgt zum Beispiel das Lernen mathematischer Fertigkeiten meist nicht mittels Auswendiglernen. Beim Addieren natürlicher Zahlen etwa ist dies gar nicht möglich, denn für jeden der beiden Summanden der Summe $x + y$ gibt es unendlich viele Werte. Für jede Kombination der beiden Werte x und y müsste in einer Liste das Tripel $(x, y, x+y)$ gespeichert werden, was unmöglich ist. Erst recht unmöglich ist dies bei Dezimalzahlen. Es stellt sich die Frage: Wie lernen wir Mathematik? Die Antwort lautet: Der Lehrer erklärt das Verfahren und die Schüler üben es an Beispielen so lange, bis sie an neuen Beispielen keine Fehler mehr machen. Nach 50 Beispielen hat der Schüler dann das Addieren (hoffentlich) verstanden. Das heißt, er kann das an nur 50 Beispielen Gelernte auf unendlich viele neue Beispiele anwenden, die er bisher noch nicht gesehen hat. Diesen Vorgang nennt man **Generalisierung**. Starten wir mit einem einfachen Beispiel.

Beispiel 8.1

Bei einem Obstbauer sollen die geernteten Äpfel automatisch in die Handelsklassen A und B eingeteilt werden. Die Sortieranlage misst mit Sensoren für jeden Apfel die zwei **Merkmale** (engl. features) Größe und Farbe und soll dann entscheiden, in welche der beiden Klassen der Apfel gehört. Dies ist eine typische **Klassifikationsaufgabe**. Systeme, welche in der Lage sind, Merkmalsvektoren in eine endliche Anzahl von Klassen einzuteilen, werden **Klassifizierer** (engl. Classifier) genannt.

Zur Einstellung der Maschine werden Äpfel von einem Fachmann handverlesen, das heißt klassifiziert. Dann werden die beiden Messwerte zusammen mit der Klasse in eine

Tab. 8.1 Trainingsdaten für den Apfelsortieragenten

Größe [cm]	8	8	6	3	...
Farbe	0,1	0,3	0,9	0,8	...
Handelsklasse	B	A	A	B	...

Tabelle (Tab. 8.1) eingetragen. Die Größe ist in Form des Durchmessers in Zentimetern angegeben und die Farbe durch einen Zahlenwert zwischen 0 (für grün) und 1 (für rot). Eine anschauliche Darstellung der Daten als Punkte in einem Streudiagramm ist in Abb. 8.2 rechts dargestellt.

Die Aufgabe beim maschinellen Lernen besteht nun darin, aus den gesammelten klassifizierten Daten eine Funktion zu generieren, die für neue Äpfel aus den beiden Merkmalen Größe und Farbe den Wert der Klasse (A oder B) berechnet. In Abb. 8.3 ist durch die in das Diagramm eingezeichnete Trennlinie solch eine Funktion dargestellt. Alle Äpfel mit einem Merkmalsvektor links unterhalb der Linie werden in die Klasse B eingeteilt und alle anderen in die Klasse A.

In diesem Beispiel ist es noch recht einfach, solch eine Trennlinie für die beiden Klassen zu finden. Deutlich schwieriger, vor allem aber weniger anschaulich, wird die Aufgabe, wenn die zu klassifizierenden Objekte nicht nur durch zwei, sondern durch viele Merkmale beschrieben werden. In der Praxis werden durchaus 30 oder mehr Merkmale verwendet. Bei n Merkmalen besteht die Aufgabe darin, in dem n -dimensionalen **Merkmalsraum** eine $(n-1)$ -dimensionale Hyperfläche zu finden, welche die beiden Klassen möglichst gut trennt. „Gut“ trennen heißt, dass der relative Anteil falsch klassifizierter Objekte möglichst klein sein soll.

Ein Klassifizierer bildet einen Merkmalsvektor auf einen Klassenwert ab. Hierbei hat er eine feste, meist kleine, Anzahl von Alternativen. Diese Abbildung wird auch **Zielfunktion**

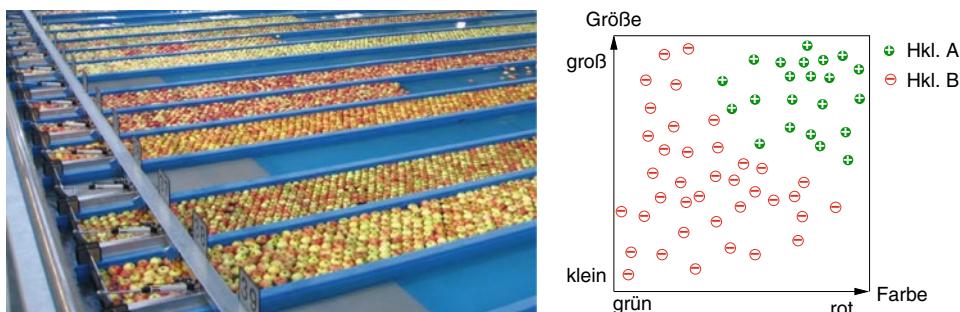
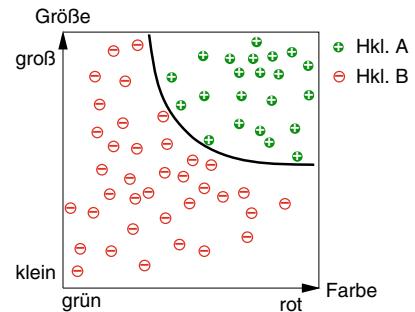


Abb. 8.2 Apfelsortieranlage der BayWa in Kressbronn und einige im Merkmalsraum klassifizierte Äpfel der Handelsklassen A und B (Foto: BayWa)

Abb. 8.3 Die im Diagramm eingezeichnete Kurve trennt die Klassen und kann dann auf beliebige neue Äpfel angewendet werden



genannt. Bildet die Zielfunktion nicht wie bei der Klassifikation auf einen endlichen Wertebereich ab, sondern auf reelle Zahlen, so handelt es sich um ein **Approximationsproblem**. Die Bestimmung des Kurses einer Aktie aus gegebenen Merkmalen ist solch ein Approximationsproblem. Wir werden im Folgenden für beide Arten von Abbildungen mehrere lernende Agenten vorstellen.

Der lernende Agent Formal können wir einen lernenden Agenten beschreiben als eine Funktion, die einen Merkmalsvektor auf einen diskreten Klassenwert oder allgemeiner auf eine reelle Zahl abbildet. Diese Funktion wird nicht programmiert, sondern sie entsteht, beziehungsweise verändert sich während der **Lernphase** unter dem Einfluss der **Trainingsdaten**. In Abb. 8.4 ist solch ein Agent dargestellt für das Beispiel der Apfelsortierung. Während des Lernens wird der Agent mit den schon klassifizierten Daten aus Tab. 8.1 gespeist. Danach stellt der Agent eine möglichst gute Abbildung des Merkmalsvektors auf den Funktionswert (z. B. Handelsklasse) dar.

Nun können wir versuchen, uns einer Definition des Begriffs „Maschinelles Lernen“ zu nähern. Tom Mitchell [Mit97] definiert:

Machine Learning is the study of computer algorithms that improve automatically through experience.

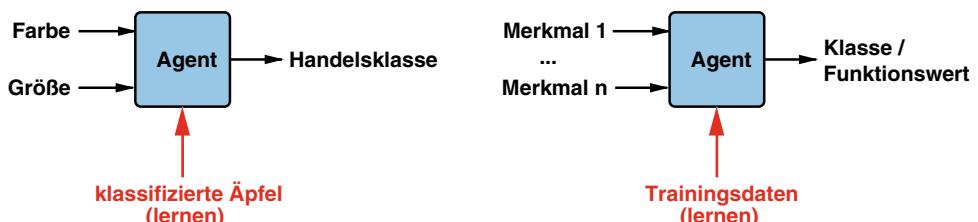


Abb. 8.4 Funktionale Struktur des lernenden Agenten für die Apfelsortierung (*links*) und allgemein (*rechts*)

In Anlehnung daran definieren wir nun:

Definition 8.1

Ein Agent heißt lernfähig, wenn sich seine Leistungsfähigkeit auf neuen, unbekannten Daten im Laufe der Zeit (nachdem er viele Trainingsbeispiele gesehen hat) verbessert (gemessen auf einem geeigneten Maßstab).

Hierbei ist es wichtig, dass die Generalisierungsfähigkeit des Lernverfahrens auf unbekannten Daten, den **Testdaten**, getestet wird. Andernfalls würde jedes System, welches die Trainingsdaten nur abspeichert, optimal abschneiden, denn beim Test auf den Trainingsdaten müssten nur die gespeicherten Daten abgerufen werden. Ein lernfähiger Agent kann charakterisiert werden durch folgende Begriffe:

Aufgabe des Lernverfahrens ist es, eine Abbildung zu lernen. Dies kann zum Beispiel die Abbildung von Größe und Farbe eines Apfels auf die Handelsklasse sein, aber auch die Abbildung von 15 Symptomen eines Patienten auf die Entscheidung, ob sein Blinddarm entfernt werden soll oder nicht.

Variabler Agent (genauer eine Klasse von Agenten): Hier wird festgelegt, mit welchem Lernverfahren gearbeitet wird. Ist dieses ausgewählt, so ist damit auch schon die Klasse aller lernbaren Funktionen bestimmt.

Trainingsdaten (Erfahrung): In den Trainingsdaten steckt das Wissen, welches von dem Lernverfahren extrahiert und gelernt werden soll. Bei der Auswahl der Trainingsdaten wie auch der Testdaten ist darauf zu achten, dass sie eine repräsentative Stichprobe für die zu lernende Aufgabe darstellen.

Testdaten sind wichtig um zu evaluieren, ob der trainierte Agent gut von den gelernten auf neue Daten generalisieren kann.

Leistungsmaß ist bei der Apfelsonderanlage die Zahl der korrekt klassifizierten Äpfel. Es wird benötigt, um die Qualität eines Agenten zu testen. Die Kenntnis des Leistungsmaßes ist meist viel schwächer als die Kenntnis der Funktion des Agenten. Es ist zum Beispiel ganz einfach, die Leistung (Zeit) eines 10.000-Meter-Läufers zu messen. Daraus folgt aber noch lange nicht, dass ein Schiedsrichter, der die Zeit misst, gleich schnell laufen kann. Der Schiedsrichter kennt eben nur das Maß, aber nicht die „Funktion“, deren Leistung er misst.

Was ist Data Mining? Die Aufgabe einer lernenden Maschine ist das Extrahieren von Wissen aus Trainingsdaten. Oft will der Entwickler oder der Anwender einer lernenden Maschine das extrahierte Wissen auch für Menschen verständlich lesbar machen. Noch besser ist es, wenn der Entwickler dieses Wissen sogar verändern kann. Die in Abschn. 8.4 vorgestellte Induktion von Entscheidungsbäumen stellt zum Beispiel eine derartige Methode dar.

Abb. 8.5 Data Mining

Aus der Wirtschaftsinformatik und dem Wissensmanagement kommen ganz ähnliche Anforderungen. Eine klassische Frage, die sich hier stellt, ist folgende: Der Betreiber eines Internetshops möchte aus der Statistik über die Aktionen der Besucher auf seinem Portal einen Zusammenhang herstellen zwischen den Eigenschaften des Kunden und der für ihn interessanten Klasse von Produkten. Dann kann der Anbieter nämlich kundenspezifisch werben. In beispielhafter Art und Weise wird uns dies demonstriert bei <http://www.amazon.de>. Dem Kunden werden zum Beispiel Produkte vorgeschlagen, die ähnlich sind zu denen, die er sich in letzter Zeit angesehen hat. In vielen Bereichen der Werbung und des Marketing sowie beim „Customer Relationship Management (CRM)“ kommen heute Data Mining Techniken zur Anwendung. Immer dann, wenn große Datenmengen vorliegen, kann versucht werden, diese für die Analyse von Kundenwünschen zu verwenden, um kundenspezifisch zu werben.

- ▶ Der Prozess des Gewinnens von Wissen aus Daten sowie dessen Darstellung und Anwendung wird als **Data Mining** bezeichnet. Die verwendeten Methoden kommen meist aus der Statistik oder dem maschinellen Lernen und sollten auch auf sehr große Datenmengen mit vertretbarem Aufwand anwendbar sein.

Im Kontext der Informationsbeschaffung, zum Beispiel im Internet oder im Intranet, spielt auch das Text Mining eine immer wichtigere Rolle. Typische Aufgaben sind das Finden ähnlicher Texte in Suchmaschinen oder die Klassifikation von Texten, wie sie zum Beispiel von Spam-Filters für Email angewendet wird. In Abschn. 8.7.1 werden wir das weit verbreitete Naive-Bayes-Verfahren zur Klassifikation von Texten einführen. Eine relativ junge Herausforderung im Data Mining ist die Gewinnung von struktureller Information aus Graphen wie sie zum Beispiel in sozialen Netzen, Verkehrsnetzen oder im Internetverkehr auftreten.

Da die beiden beschriebenen Aufgabenstellungen des maschinellen Lernens und des Data Mining formal sehr ähnlich sind, sind die in beiden Gebieten verwendeten Metho-

den im Wesentlichen identisch. Daher wird bei der Beschreibung der Lernverfahren nicht zwischen maschinellem Lernen und Data Mining unterschieden.

Durch die große kommerzielle Bedeutung von Data Mining-Techniken gibt es heute eine ganze Reihe mächtiger Data Mining-Systeme, die dem Anwender für alle zur Extraktion des Wissens aus Daten benötigten Aufgaben eine große Palette von komfortablen Werkzeugen bieten. In Abschn. 8.10 werden wir solch ein System vorstellen.

8.1 Datenanalyse

Die Statistik stellt eine ganze Auswahl von Methoden bereit, um Daten durch einfache Kenngrößen zu beschreiben. Wir wählen daraus einige für die Analyse von Trainingsdaten besonders wichtige aus und testen diese am Beispiel einer Teilmenge der LEXMED-Daten aus Abschn. 7.3. In dieser Datenmenge sind für $N = 473$ Patienten die in Tab. 8.2 kurz beschriebenen Symptome x_1, \dots, x_{15} sowie die Diagnose x_{16} (Appendizitis negativ/positiv) eingetragen. Der Patient Nummer eins wird zum Beispiel durch den Vektor

$$\mathbf{x}^1 = (26, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 37, 9, 38, 8, 23, 100, 0, 1)$$

und Patient Nummer zwei durch

$$\mathbf{x}^2 = (17, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 36, 9, 37, 4, 8100, 0, 0)$$

beschrieben. Patient Nummer zwei hat den Leukozytenwert $x_{14}^2 = 8100$.

Für jede Variable x_i ist ihr Mittelwert \bar{x}_i definiert als

$$\bar{x}_i := \frac{1}{N} \sum_{p=1}^N x_i^p$$

und die Standardabweichung s_i als Maß für ihre mittlere Abweichung vom Mittelwert als

$$s_i := \sqrt{\frac{1}{N-1} \sum_{p=1}^N (x_i^p - \bar{x}_i)^2}.$$

Wichtig für die Analyse von mehrdimensionalen Daten ist die Frage, ob zwei Variablen x_i und x_j statistisch abhängig (korreliert) sind. Hierüber gibt z. B. die Kovarianz

$$\sigma_{ij} = \frac{1}{N-1} \sum_{p=1}^N (x_i^p - \bar{x}_i)(x_j^p - \bar{x}_j)$$

Auskunft. In dieser Summe liefert der Summand für den p -ten Datenvektor genau dann einen positiven Beitrag, wenn seine i -te und j -te Komponente beide nach oben oder beide

Tab. 8.2 Beschreibung der Variablen x_1, \dots, x_{16}

Var.-Nr.	Beschreibung	Werte
1	Alter	stetig
2	Geschlecht (1 = männl., 2 = weibl.)	1, 2
3	Schmerz Quadrant 1	0, 1
4	Schmerz Quadrant 2	0, 1
5	Schmerz Quadrant 3	0, 1
6	Schmerz Quadrant 4	0, 1
7	Lokale Abwehrspannung	0, 1
8	Generalisierte Abwehrspannung	0, 1
9	Schmerz bei Loslassmanöver	0, 1
10	Erschütterung	0, 1
11	Schmerz bei rektaler Untersuchung	0, 1
12	Temperatur axial	stetig
13	Temperatur rektal	stetig
14	Leukozyten	stetig
15	Diabetes mellitus	0, 1
16	Appendizitis	0, 1

nach unten vom Mittelwert abweichen. Haben die Abweichungen unterschiedliche Vorzeichen, so ergibt sich ein negativer Beitrag zur Summe. Daher sollte zum Beispiel die Kovarianz $\sigma_{12,13}$ der beiden verschiedenen Fieberwerte ganz deutlich positiv sein.

Die Kovarianz hängt allerdings auch noch vom absoluten Wert der Variablen ab, was den Vergleich der Werte schwierig macht. Um im Falle von mehreren Variablen den Grad der Abhängigkeit von Variablen untereinander vergleichen zu können, definiert man daher für zwei Variablen x_i und x_j den **Korrelationskoeffizienten**

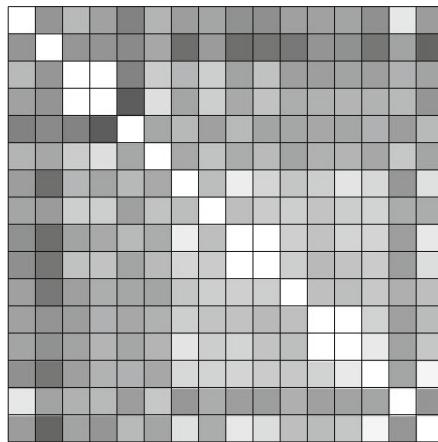
$$K_{ij} = \frac{\sigma_{ij}}{s_i \cdot s_j}$$

als normierte Kovarianz. Die Matrix K aller Korrelationskoeffizienten enthält Werte zwischen -1 und 1 , ist symmetrisch, und alle Diagonalelemente haben den Wert 1 . In Tab. 8.3 ist die Korrelationsmatrix für alle 16 Variablen angegeben.

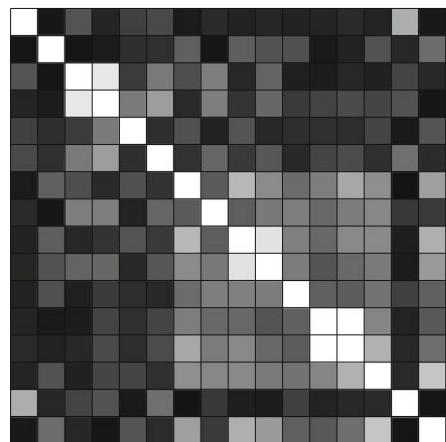
Etwas anschaulicher wird diese Matrix, wenn man sie als Dichteplot darstellt. Statt den numerischen Werten der Matrixelemente sind in Abb. 8.6 die Felder der Matrix mit Grauwerten gefüllt. Im rechten Diagramm sind die Beträge der Werte dargestellt. So erkennt man sehr schnell, welche Variablen eine schwache oder starke Abhängigkeit aufweisen. Man sieht zum Beispiel, dass die Variablen 7, 9, 10 und 14 die stärkste Korrelation mit der Klassenvariable *Appendizitis* aufweisen und daher für die Diagnose wichtiger als die anderen Variablen sind. Man erkennt aber auch, dass die Variablen 9 und 10 stark korreliert sind. Dies könnte bedeuten, dass für die Diagnose eventuell einer der beiden Werte genügt.

Tab. 8.3 Korrelationsmatrix für die 16 Appendizitis-Variablen, gemessen auf 473 Fällen

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



$$K_{ij} = \begin{cases} 1: \text{schwarz}, & K_{ij} = 1: \text{wei\ss} \end{cases}$$



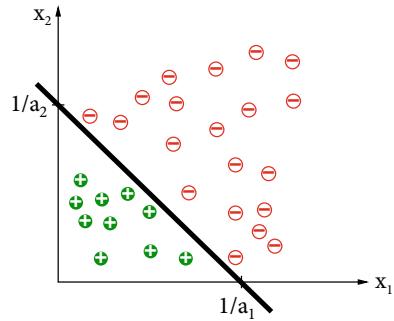
$$K_{ij} = 0: \text{schwarz}, \quad K_{ij} = 1: \text{wei\ss}$$

Abb. 8.6 Die Korrelationsmatrix als Dichteplot. Im *linken Diagramm* steht *dunkel* für negativ und *hell* für positiv. Im *rechten Bild* wurden die Absolutbeträge eingetragen. Hier bedeutet *schwarz* $K_{ij} \approx 0$ (unkorrielt) und *wei\ss* $|K_{ij}| \approx 1$ (starke Korrelation)

8.2 Das Perzeptron, ein linearer Klassifizierer

Im Klassifikationsbeispiel der Apfelsortierung ist in Abb. 8.3 eine gekrümmte Trennlinie zur Trennung der beiden Klassen eingezeichnet. Ein einfacherer Fall ist in Abb. 8.7 dargestellt. Hier lassen sich die zweidimensionalen Trainingsbeispiele durch eine Gerade trennen. Man nennt solch eine Menge von Trainingsdaten dann **linear separabel**. In n Dimensionen wird zur Trennung eine Hyperebene benötigt. Diese stellt einen linearen Unterraum der Dimension $n - 1$ dar.

Abb. 8.7 Eine linear separable zweidimensionale Datenmenge. Die Trenngerade hat die Gleichung $a_1x_1 + a_2x_2 = 1$



Da sich jede $(n - 1)$ -dimensionale Hyperebene im \mathbb{R}^n durch eine Gleichung

$$\sum_{i=1}^n a_i x_i = \theta$$

beschreiben lässt, ist es sinnvoll, die lineare Separabilität wie folgt zu definieren.

Definition 8.2

Zwei Mengen $M_1 \subset \mathbb{R}^n$ und $M_2 \subset \mathbb{R}^n$ heißen **linear separabel**, wenn reelle Zahlen a_1, \dots, a_n, θ existieren mit

$$\sum_{i=1}^n a_i x_i > \theta \quad \text{für alle } \mathbf{x} \in M_1 \quad \text{und} \quad \sum_{i=1}^n a_i x_i \leq \theta \quad \text{für alle } \mathbf{x} \in M_2.$$

Der Wert θ wird als Schwelle bezeichnet.

In Abb. 8.8 erkennen wir, dass die AND-Funktion linear separabel ist, die XOR-Funktion hingegen nicht. Bei AND trennt zum Beispiel die Gerade $-x_1 + 3/2$ wahre und falsche Belegungen der Formel $x_1 \wedge x_2$. Bei XOR hingegen gibt es keine Trenngerade. Offenbar besitzt die XOR-Funktion diesbezüglich eine komplexere Struktur als die AND-Funktion.

Wir stellen nun mit dem Perzeptron ein ganz einfaches Lernverfahren vor, das genau die linear separablen Mengen trennen kann.

Definition 8.3

Sei $\mathbf{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$ ein Gewichtsvektor und $\mathbf{x} \in \mathbb{R}^n$ ein Eingabevektor. Ein **Perzeptron** stellt eine Funktion $P : \mathbb{R}^n \rightarrow \{0, 1\}$ dar, die folgender Regel entspricht:

$$P(\mathbf{x}) = \begin{cases} 1 & \text{falls } \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^n w_i x_i > 0 \\ 0 & \text{sonst} \end{cases}$$

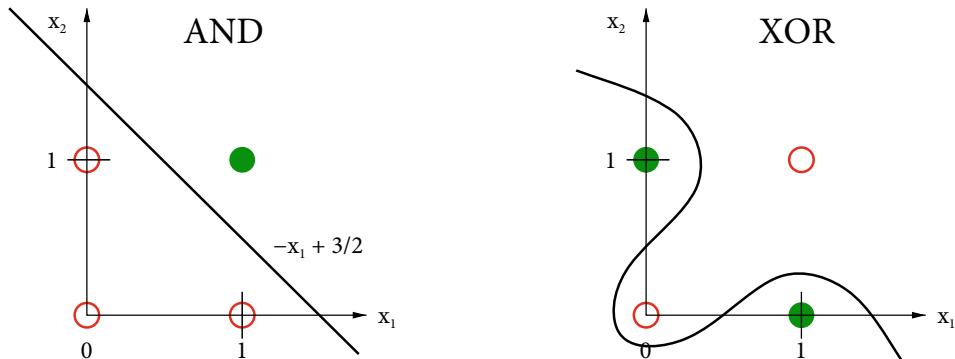
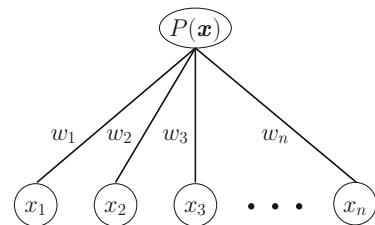


Abb. 8.8 Die boolesche Funktion AND ist linear separabel, XOR hingegen nicht ($\bullet \hat{=} \text{ wahr}$, $\circ \hat{=} \text{ falsch}$)

Abb. 8.9 Graphische Darstellung eines Perzeptrons als zweilagiges neuronales Netz



Das Perzeptron [Ros58, MP69] ist ein sehr einfacher Klassifikationsalgorithmus. Es ist äquivalent zu einem zweilagigen gerichteten neuronalen Netzwerk mit Aktivierung durch eine Schwellwertfunktion, dargestellt in Abb. 8.9. Wie in Kap. 9 dargestellt, steht jeder Knoten in dem Netz für ein Neuron und jede Kante für eine Synapse. Vorläufig betrachten wir das Perzeptron jedoch nur als lernenden Agenten, das heißt als eine mathematische Funktion, die einen Merkmalsvektor auf einen Funktionswert abbildet. Die EingabevARIABLEN x_i werden hier als **Merkmaile** (engl. features) bezeichnet.

Wie man an der Formel $\sum_{i=1}^n w_i x_i > 0$ erkennt, werden alle Punkte \mathbf{x} über der Hyperebene $\sum_{i=1}^n w_i x_i = 0$ als positiv ($P(\mathbf{x}) = 1$) klassifiziert. Alle anderen als negativ ($P(\mathbf{x}) = 0$). Die trennende Hyperebene geht durch den Ursprung, denn es gilt $\theta = 0$. Dass das Fehlen einer beliebigen Schwelle keine Einschränkung der Mächtigkeit darstellt, werden wir über einen kleinen Trick noch zeigen. Zuerst aber wollen wir einen einfachen Lernalgorithmus für das Perzeptron vorstellen.

8.2.1 Die Lernregel

Mit den Bezeichnungen M_+ und M_- für die Mengen der positiven, bzw. negativen Trainingsmuster lautet die Perzeptron-Lernregel [MP69]:

PERZEPTRONLERNEN(M_+, M_-)

w = beliebiger Vektor reeller Zahlen

Repeat

For all $x \in M_+$

If $w \cdot x \leq 0$ **Then** $w = w + x$

For all $x \in M_-$

If $w \cdot x > 0$ **Then** $w = w - x$

Until alle $x \in M_+ \cup M_-$ werden korrekt klassifiziert

Das Perzeptron soll für alle $x \in M_+$ den Wert 1 ausgeben. Nach Definition 8.3 muss dazu $w \cdot x > 0$ sein. Ist dies nicht der Fall, so wird x zum Gewichtsvektor w addiert, wodurch der Gewichtsvektor genau in der richtigen Richtung geändert wird. Das erkennt man, wenn man das Perzeptron auf den geänderten Vektor $w + x$ anwendet, denn

$$(w + x) \cdot x = w \cdot x + x^2.$$

Wird dieser Schritt oft genug wiederholt, so wird irgendwann der Wert $w \cdot x$ positiv, wie es sein soll. Analog sieht man, dass für negative Trainingsdaten das Perzeptron einen immer kleineren Wert

$$(w - x) \cdot x = w \cdot x - x^2$$

berechnet, der irgendwann negativ wird.²

Beispiel 8.2

Ein Perzeptron soll auf den Mengen $M_+ = \{(0, 1, 8), (2, 0, 6)\}$ und $M_- = \{(-1, 2, 1, 4), (0, 4, -1)\}$ trainiert werden. Als initialer Gewichtsvektor wird $w = (1, 1)$ verwendet. Die Trainingsdaten und die durch den Gewichtsvektor definierte Gerade $w \cdot x = x_1 + x_2 = 0$ sind in Abb. 8.10 im ersten Bild der oberen Reihe dargestellt. Gestrichelt eingezeichnet ist außerdem der Gewichtsvektor. Wegen $w \cdot x = 0$ steht dieser senkrecht auf der Geraden.

Beim ersten Durchlauf der Schleife des Lernalgorithmus ist $(-1, 2, 1, 4)$ das einzige falsch klassifizierte Trainingsbeispiel, denn

$$(-1, 2, 1, 4) \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0,2 > 0.$$

Also ergibt sich $w = (1, 1) - (-1, 2, 1, 4) = (2, 2, -0, 4)$, wie im zweiten Bild der oberen Reihe in Abb. 8.10 eingezeichnet. Die weiteren Bilder zeigen, wie nach insgesamt fünf

² Vorsicht! Dies ist kein Konvergenzbeweis für die Perzeptron Lernregel. Es zeigt nur, dass das Perzeptron konvergiert, wenn die Trainingsdatensetze aus einem einzigen Beispiel besteht.

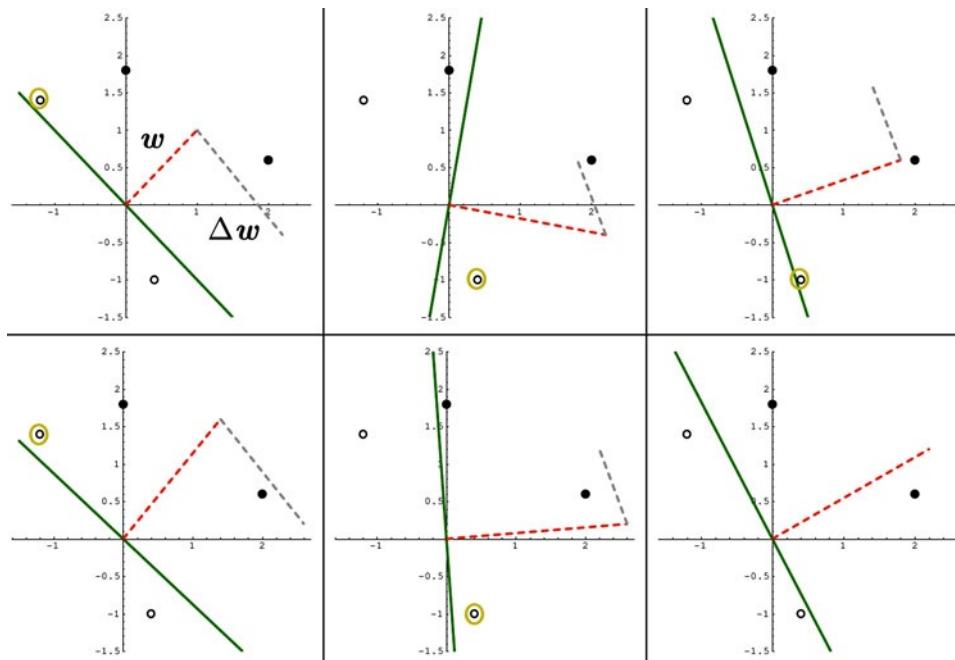


Abb. 8.10 Anwendung der Perzeptronlernregel auf zwei positive (\bullet) und zwei negative (\circ) Datenpunkte. Die durchgezogene Linie stellt die aktuelle Trennerade $w x = 0$ dar. Die senkrecht darauf stehende gestrichelte Linie ist der Gewichtsvektor w und die zweite gestrichelte Linie der zu w zu addierende Änderungsvektor $\Delta w = x$ oder $\Delta w = -x$, der sich aus dem jeweils aktiven grau umrandeten Datenpunkt berechnet

Änderungen die Trennerade zwischen den beiden Klassen liegt. Das Perzeptron klassifiziert nun also alle Daten korrekt. Man erkennt an dem Beispiel schön, dass jeder falsch klassifizierte Datenpunkt aus M_+ den Gewichtsvektor w in seine Richtung „zieht“ und dass jeder falsch klassifizierte Punkt aus M_- den Gewichtsvektor in die Gegenrichtung „schiebt“.

Dass das Perzeptron bei linear separablen Daten immer konvergiert, wurde in [MP69] gezeigt. Es gilt:

Satz 8.1

Es seien die Klassen M_+ und M_- linear separabel durch eine Hyperebene $w x = 0$. Dann konvergiert PERZEPTRONLERNEN für jede Initialisierung ($\neq 0$) des Vektors w . Das Perzeptron P mit dem so berechneten Gewichtsvektor trennt die Klassen M_+

und M_- , d. h.

$$P(\mathbf{x}) = 1 \iff \mathbf{x} \in M_+$$

und

$$P(\mathbf{x}) = 0 \iff \mathbf{x} \in M_-.$$

Wie man an dem Beispiel 8.2 schön sieht, kann das Perzeptron wie oben definiert nicht beliebige linear separable Mengen trennen, sondern nur solche, die durch eine Ursprungsgerade, bzw. im \mathbb{R}^n durch eine Hyperebene im Ursprung trennbar sind, denn es fehlt der konstante Term θ in der Gleichung $\sum_{i=1}^n w_i x_i = 0$.

Durch folgenden Trick kann man den konstanten Term erzeugen. Man hält die letzte Komponente x_n des Eingabevektors \mathbf{x} fest und setzt sie auf den Wert 1. Nun wirkt das Gewicht $w_n =: -\theta$ wie eine Schwelle, denn es gilt

$$\sum_{i=1}^n w_i x_i = \sum_{i=1}^{n-1} w_i x_i - \theta > 0 \iff \sum_{i=1}^{n-1} w_i x_i > \theta.$$

Solch ein konstanter Wert $x_n = 1$ in der Eingabe wird im Englischen als **bias unit** bezeichnet. Da das zugehörige Gewicht eine feste Verschiebung der Hyperebene erzeugt, ist die Bezeichnung „bias“ (Verschiebung) passend.

Bei der Anwendung des Perzeptron-Lernalgorithmus wird nun an jeden Trainingsdatenvektor ein Bit mit dem konstanten Wert 1 angehängt. Man beachte, dass beim Lernen auch das Gewicht w_n , bzw. die Schwelle θ gelernt wird.

Nun ist also gezeigt, dass ein Perzeptron $P_\theta : \mathbb{R}^{n-1} \rightarrow \{0, 1\}$

$$P_\theta(x_1, \dots, x_{n-1}) = \begin{cases} 1 & \text{falls } \sum_{i=1}^{n-1} w_i x_i > \theta \\ 0 & \text{sonst} \end{cases} \quad (8.1)$$

mit beliebiger Schwelle durch ein Perzeptron $P : \mathbb{R}^n \rightarrow \{0, 1\}$ mit Schwelle 0 simuliert werden kann. Zusammenfassend können wir also festhalten:

Satz 8.2

Eine Funktion $f : \mathbb{R}^n \rightarrow \{0, 1\}$ kann von einem Perzeptron genau dann dargestellt werden, wenn die beiden Mengen der positiven und negativen Eingabevektoren linear separabel sind.

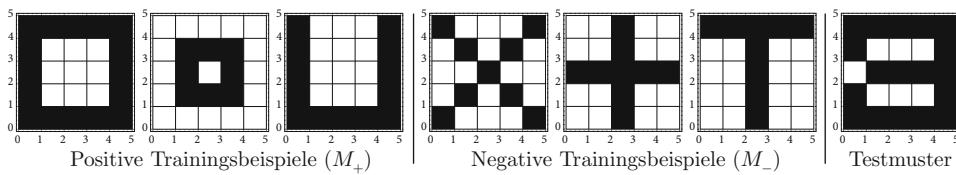
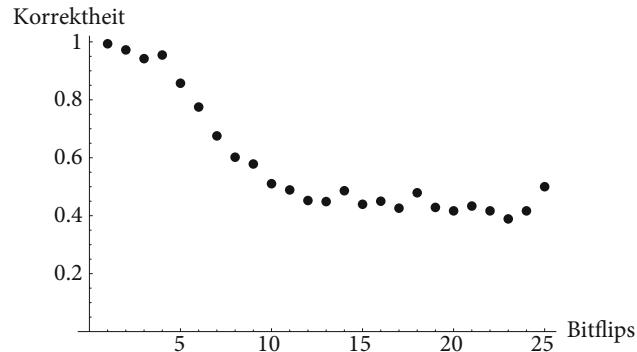


Abb. 8.11 Die sechs verwendeten Trainingsmuster. Das *ganz rechte* Muster ist eines der 22 Testmuster für das erste Muster mit 4 aufeinander folgenden invertierten Bits

Abb. 8.12 Relative Korrektheit des Perzeptrons in Abhängigkeit von der Zahl invertierter Bits in den Testdaten



Beispiel 8.3

Wir trainieren nun ein Perzeptron mit Schwelle auf sechs einfachen graphischen, in Abb. 8.11 dargestellten binären Mustern mit je 5×5 Pixeln.

Die Trainingsdaten werden von PERZEPTRONLERNEN in 4 Iterationen über alle Muster gelernt. Zum Testen der Generalisierungsfähigkeit wurden verrauschte Muster mit einer variablen Anzahl invertierter Bits verwendet. Die invertierten Bits in den Testmustern sind jeweils aufeinander folgend. In Abb. 8.12 ist der Prozentsatz korrekt erkannter Muster in Abhängigkeit von der Anzahl falscher Bits dargestellt.

Ab etwa fünf aufeinander folgenden invertierten Bits geht die Korrektheit stark zurück, was bei der Einfachheit des Modells nicht überraschend ist. Im nächsten Abschnitt werden wir ein Verfahren vorstellen, das hier deutlich besser abschneidet.

8.2.2 Optimierung und Ausblick

Als eines der einfachsten neuronalen Lernverfahren kann das zweilagige Perzeptron nur linear separable Klassen trennen. In Abschn. 9.5 werden wir sehen, dass mehrlagige Netze wesentlich mächtiger sind. Trotz der einfachen Struktur konvergiert das Perzeptron in der vorgestellten Form nur sehr langsam. Eine Beschleunigung kann erreicht werden durch Normierung der Gewichtsänderungsvektoren. Die Formeln $w = w \pm x$ werden ersetzt durch

$w = w \pm x / |x|$. Dadurch hat jeder Datenpunkt das gleiche Gewicht beim Lernen, unabhängig von seinem Betrag.

Die Konvergenzgeschwindigkeit hängt beim Perzepron stark ab von der Initialisierung des Vektors w . Im Idealfall muss er gar nicht mehr geändert werden und der Algorithmus konvergiert nach einer Iteration. Diesem Ziel kann man etwas näherkommen durch die heuristische Initialisierung

$$w_0 = \sum_{x \in M_+} x - \sum_{x \in M_-} x,$$

die in Aufgabe 8.5 genauer untersucht werden soll.

Vergleicht man die Formel des Perzeprons mit den in Abschn. 7.3.1 vorgestellten Scores, so erkennt man sofort deren Äquivalenz. Außerdem ist das Perzepron als einfaches neuronales Netzwerkmodell äquivalent zu Naive-Bayes, dem einfachsten Typ von Bayes-Netz (siehe Aufgabe 8.17). So haben offenbar mehrere sehr unterschiedliche Klassifikationsverfahren einen gemeinsamen Ausgangspunkt.

In Kap. 9 werden wir in Form des Backpropagation-Algorithmus eine Verallgemeinerung des Perzeprons kennenlernen, die unter anderem durch die Verwendung mehrerer Schichten auch nicht linear separable Mengen trennen kann und eine verbesserte Lernregel besitzt.

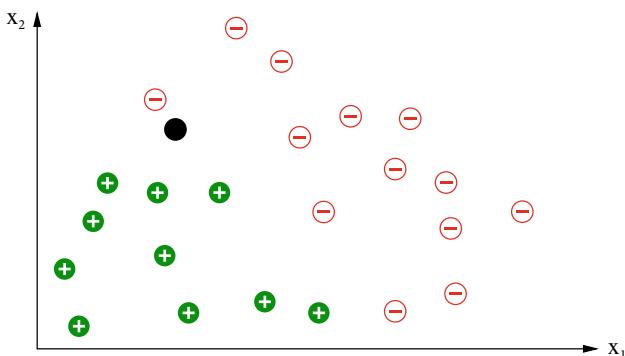
8.3 Nearest Neighbour-Methoden

Beim Perzepron wird das in den Trainingsdaten vorhandene Wissen extrahiert und in komprimierter Form in den Gewichten w_i gespeichert. Hierbei geht Information über die Daten verloren. Genau dies ist aber erwünscht, denn das System soll von den Trainingsdaten auf neue Daten generalisieren. Das Generalisieren ist ein unter Umständen sehr zeitaufwändiger Prozess mit dem Ziel, eine kompakte Repräsentation der Daten in Form einer Funktion zu finden, die neue Daten möglichst gut klassifiziert.

Ganz im Gegensatz dazu werden beim Auswendiglernen alle Daten einfach abgespeichert. Das Lernen ist hier also ganz einfach. Wie schon erwähnt, ist das nur gespeicherte Wissen aber nicht so einfach auf neue unbekannte Beispiele anwendbar. Sehr unpassend ist zum Beispiel solch ein Vorgehen, um das Skifahren zu erlernen. Nur durch das Anschauen von Videos guter Skifahrer wird ein Anfänger nie ein guter Skifahrer werden. Offenbar passiert beim Lernen von derartigen automatisierten Bewegungsabläufen etwas ähnliches wie beim Perzepron. Nach genügend langem Üben wird das in den Trainingsbeispielen gespeicherte Wissen transformiert in eine interne Repräsentation im Gehirn.

Es gibt aber durchaus erfolgreiche Beispiele des Auswendiglernens, bei denen auch Generalisierung möglich ist. Ein Arzt etwa könnte versuchen, sich bei der Diagnose eines schwierigen Falles an ähnlich gelagerte Fälle aus der Vergangenheit zu erinnern. Falls er in seinem Gedächtnis fündig wird, so schlägt er in seinen Akten diesen Fall nach und

Abb. 8.13 In diesem Beispiel mit negativen und positiven Trainingsbeispielen weist die Nearest Neighbour-Methode dem neuen schwarz markierten Punkt die negative Klasse zu



wird dann gegebenenfalls die gleiche oder eine ähnliche Diagnose stellen. Bei dieser Vorgehensweise ergeben sich zwei Schwierigkeiten. Erstens muss der Arzt ein gutes Gefühl für **Ähnlichkeit** besitzen, um sich an den ähnlichsten Fall zu erinnern. Hat er diesen gefunden, so stellt sich die Frage, ob dieser ähnlich genug ist, um die gleiche Diagnose zu rechtfertigen.

Was bedeutet Ähnlichkeit in dem hier aufgebauten formalen Kontext? Wir repräsentieren die Trainingsbeispiele wie gewohnt in einem mehrdimensionalen Merkmalsraum und definieren: *Zwei Beispiele sind umso ähnlicher, je geringer ihr Abstand im Merkmalsraum ist.*

Diese Definition wenden wir an auf das einfache zweidimensionale Beispiel aus Abb. 8.13. Hier ist der nächste Nachbar zu dem schwarzen Punkt ein negatives Beispiel. Also wird er der negativen Klasse zugeordnet.

Der Abstand $d(\mathbf{x}, \mathbf{y})$ zwischen zwei Punkten $\mathbf{x} \in \mathbb{R}^n$ und $\mathbf{y} \in \mathbb{R}^n$ kann zum Beispiel gemessen werden mit der durch die euklidische Norm gegebenen Metrik

$$d(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

Da es neben dieser Metrik noch viele andere Abstandsmaße gibt, ist es angebracht, sich für eine konkrete Anwendung Gedanken über Alternativen zu machen. In vielen Anwendungen sind bestimmte Merkmale wichtiger als andere. Es ist daher oft sinnvoll, die Merkmale durch Gewichte w_i unterschiedlich zu skalieren. Die Formel lautet dann

$$d_w(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^n w_i (x_i - y_i)^2}.$$

Das folgende einfache Programm zur **Nearest Neighbour-Klassifikation** sucht in den Trainingsdaten den nächsten Nachbarn t zu dem neuen Beispiel s und klassifiziert s dann gleich wie t .³

³ Die Funktionale argmin und argmax bestimmen, ähnlich wie min und max, Minimum oder Maximum einer Menge oder Funktion. Sie liefern aber nicht den Wert des Minimums oder Maximums,

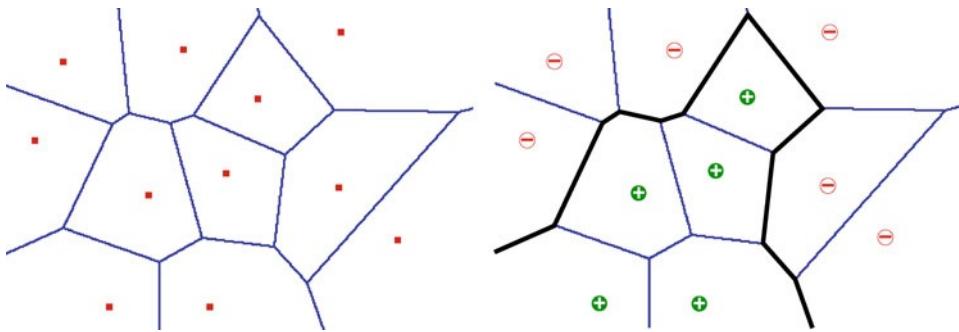


Abb. 8.14 Eine Punktmenge zusammen mit ihrem Voronoi-Diagramm (*links*) und die daraus erzeugte Linie zur Trennung der beiden Klassen M_+ und M_-

NEARESTNEIGHBOUR(M_+, M_-, s)

```

 $t = \operatorname{argmin}_{x \in M_+ \cup M_-} \{d(s, x)\}$ 
If  $t \in M_+$  Then Return(,,+")
Else Return(,,-")

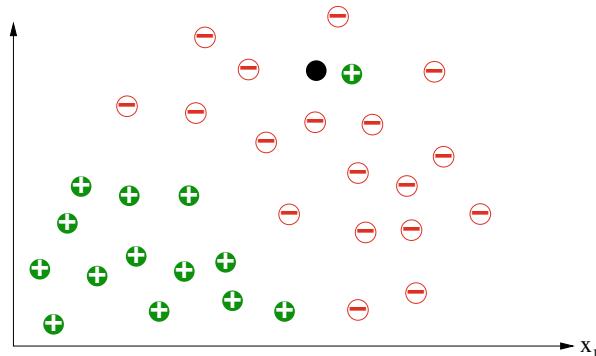
```

Im Unterschied zum Perzeptron erzeugt die Nearest Neighbour-Methode aus den Trainingsdatenpunkten keine Linie, welche die beiden Klassen trennt. Eine imaginäre Linie zur Trennung der beiden Klassen existiert aber durchaus. Diese kann man erzeugen, indem man zuerst das so genannte **Voronoi-Diagramm** erzeugt. Im Voronoi-Diagramm ist jeder Datenpunkt von einem konvexen Polygon umgeben, welches damit ein Gebiet um diesen definiert. Das Voronoi-Diagramm besitzt die Eigenschaft, dass zu einem beliebigen Punkt der nächste Nachbar unter den Datenpunkten der Datenpunkt ist, welcher im gleichen Gebiet liegt. Ist nun also für eine Menge von Trainingsdaten das zugehörige Voronoi-Diagramm bestimmt, so ist es einfach, für einen neuen, zu klassifizierenden Punkt dessen nächsten Nachbarn zu finden. Die Klassenzugehörigkeit wird dann von dem nächsten Nachbarn übernommen.

An Abb. 8.14 erkennt man schön, dass die Nearest Neighbour-Methode wesentlich mächtiger ist als das Perzeptron. Sie ist in der Lage, beliebig komplexe Trennlinien (allgemein: Hyperflächen) korrekt zu realisieren. Allerdings liegt genau darin auch eine Gefahr. Ein einziger fehlerhafter Punkt kann nämlich unter Umständen zu sehr schlechten Klassifikationsergebnissen führen. Solch ein Fall tritt in Abb. 8.15 bei der Klassifikation des schwarzen Punktes auf. Die Nearest Neighbour-Methode klassifiziert diesen eventuell falsch. Wenn nämlich der unmittelbar neben dem schwarzen Punkt liegende positive Punkt ein statistischer Ausreißer der positiven Klasse ist, so wird der Punkt positiv klassi-

sondern die Stelle, an der dieses auftritt, also das Argument und nicht den Funktionswert des Minimums oder Maximums.

Abb. 8.15 Die Nearest Neighbour-Methode weist hier dem neuen *schwarz markierten Punkt* die falsche (positive) Klasse zu, denn der nächste Nachbar ist sehr wahrscheinlich falsch klassifiziert



fizierte. Erwünscht wäre hier aber eine negative Klassifikation. Solche Fehlanpassungen an zufällige Fehler (Rauschen) nennt man **Überanpassung** (engl. overfitting).

Um Fehlklassifikationen durch einzelne Ausreißer zu verhindern, ist es empfehlenswert, die Trennflächen etwas zu glätten. Dies kann man zum Beispiel erreichen mit dem Algorithmus k -NEARESTNEIGHBOUR in Abb. 8.16, welcher einen Mehrheitsentscheid unter den k nächsten Nachbarn durchführt.

Beispiel 8.4

Wir wenden nun NEARESTNEIGHBOUR auf das Beispiel 8.3 an. Da die Daten hier binär vorliegen, bietet es sich an, als Metrik den Hamming-Abstand zu verwenden.⁴ Als Testbeispiele verwenden wir wieder modifizierte Trainingsbeispiele mit jeweils n aufeinanderfolgenden invertierten Bits. In Abb. 8.17 ist die relative Anzahl der korrekt klassifizierten Testbeispiele über der Zahl invertierter Bits b aufgetragen. Bei bis zu acht invertierten Bits werden alle Muster korrekt erkannt. Darüber nimmt der Fehler schnell zu. Dies ist nicht überraschend, denn das Trainingsmuster Nr. 2 aus Abb. 8.11 aus der Klasse M_+ hat einen Hamming-Abstand von 9 zu den beiden Trainingsmustern Nr. 4 und 5 aus der anderen Klasse. Das heißt, die Testmuster liegen mit hoher Wahrscheinlichkeit nah an Mustern der anderen Klasse. Ganz klar erkennt man, dass die Nearest Neighbour-Klassifikation dem Perzeptron bezüglich der Korrektheit im relevanten Bereich bis 8 falsche Bits deutlich überlegen ist.

8.3.1 Zwei Klassen, viele Klassen, Approximation

Die Nearest Neighbour-Klassifikation lässt sich auch auf mehr als zwei Klassen anwenden. Die Klasse des zu klassifizierenden Merkmalsvektors wird, wie auch bei zwei Klassen,

⁴ Der Hamming-Abstand zweier Bit-Vektoren ist die Anzahl unterschiedlicher Bits der beiden Vektoren.

K-NEARESTNEIGHBOUR(M_+ , M_- , s)

```

 $V = \{k \text{ nächste Nachbarn in } M_+ \cup M_-\}$ 
If  $|M_+ \cap V| > |M_- \cap V|$  Then Return(„+“)
ElseIf  $|M_+ \cap V| < |M_- \cap V|$  Then Return(„-“)
Else Return(Random(„+“, „-“))

```

Abb. 8.16 Der Algorithmus K-NEARESTNEIGHBOUR

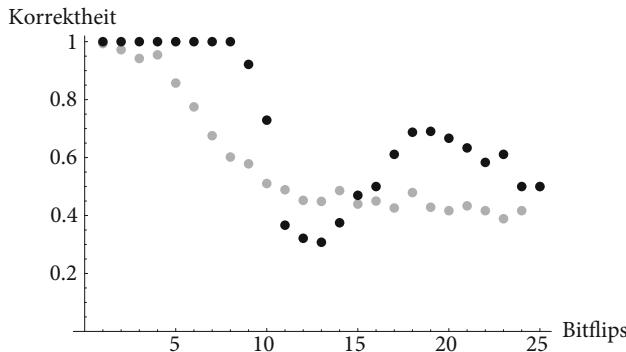


Abb. 8.17 Relative Korrektheit der Nearest Neighbour-Klassifikation in Abhängigkeit von der Zahl invertierter Bits. Die Struktur der Kurve mit dem Minimum bei 13 und Maximum bei 19 hängt mit der speziellen Struktur der Trainingsmuster zusammen. Zum Vergleich sind in grau die Werte des Perzeptrons aus Beispiel 8.3 eingezeichnet

einfach als die Klasse des nächsten Nachbarn festgelegt. Bei der k -Nearest-Neighbour-Methode wird die Klasse ermittelt als die Klasse mit den meisten Mitgliedern unter den k nächsten Nachbarn.

Wird die Zahl der Klassen groß, so ist es meist nicht mehr sinnvoll, Klassifikationsverfahren einzusetzen, denn die Zahl der benötigten Trainingsdaten wächst stark mit der Zahl der Klassen. Außerdem geht bei der Klassifikation mit vielen Klassen unter Umständen wichtige Information verloren. Dies wird an folgendem Beispiel deutlich.

Beispiel 8.5

Ein autonomer Roboter mit einfacher Sensorik ähnlich zu den in Abb. 1.1 vorgestellten Braitenberg-Vehikeln soll lernen, sich vom Licht wegzubewegen. Das bedeutet, er soll lernen, seine Sensorwerte möglichst optimal auf ein Steuersignal abzubilden, welches dem Antrieb die zu fahrende Richtung vorgibt. Der Roboter sei mit zwei einfachen Lichtsensoren auf der Vorderseite ausgestattet. Aus den beiden Sensorsignalen s_l des linken und s_r des rechten Sensors wird das Verhältnis $x = s_r/s_l$ berechnet. Zur Ansteuerung der Elektromotoren der beiden Räder soll aus diesem Wert x die Differenz $v = U_r - U_l$ der beiden Spannungen U_r am rechten und U_l am linken Motor bestimmt

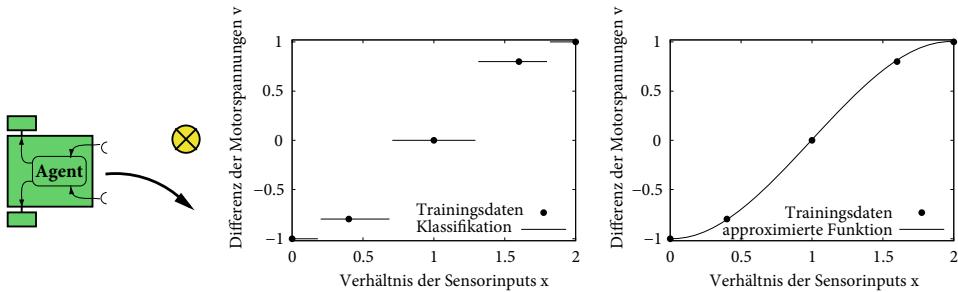


Abb. 8.18 Der lernende Agent, der dem Licht ausweichen soll (*links*), dargestellt als Classifier (*Mitte*) und als Approximation (*rechts*)

werden. Die Aufgabe des lernenden Agenten sei es nun, einem Lichtsignal auszuweichen. Er muss also eine Abbildung f finden, die für jeden Wert x den „richtigen“ Wert $v = f(x)$ berechnet.⁵

Dazu führen wir Experimente durch, indem wir für einige wenige gemessene Werte x einen möglichst optimalen Wert v ermitteln. Diese Werte sind in Abb. 8.18 einge tragen und sollen dem lernenden Agenten als Trainingsdaten dienen. Bei der Nearest Neighbour-Klassifikation wird nun jeder Punkt im Merkmalsraum (d. h. auf der x -Achse) gleich klassifiziert wie sein nächster Nachbar unter den Trainingsdaten. Die Funktion zur Ansteuerung der Motoren ist dann also eine Stufenfunktion mit großen Sprüngen (Abb. 8.18 Mitte). Will man eine feinere diskrete Abstufung haben, so muss man entsprechend mehr Trainingsdaten bereitstellen. Eine stetige Funktion erhält man hingegen, wenn man an die gegebenen fünf Punkte eine entsprechend glatte Funktion approximiert (Abb. 8.18 Mitte). Die Forderung nach der Stetigkeit der Funktion f führt hier also bei wenigen Punkten zu sehr guten Resultaten.

Für die Approximation von Funktionen an Datenpunkte gibt es eine Reihe mathematischer Methoden, etwa die Polynominterpolation, die Spline-Interpolation oder die Methode der kleinsten Quadrate. Problematisch wird die Anwendung dieser Methoden in hohen Dimensionen. Die besondere Schwierigkeit in der KI besteht darin, dass modellfreie Approximationsmethoden benötigt werden. Das heißt, ohne Wissen über spezielle Eigenschaften der Daten und der Anwendung soll eine gute Approximation an die Daten erfolgen. Sehr gute Erfolge wurden hier erzielt mit den in Kap. 9 vorgestellten neuronalen Netzen.

Die k -Nearest-Neighbour-Methode lässt sich in einfacher Weise auch auf Approximationsprobleme anwenden. In dem Algorithmus κ -NEARESTNEIGHBOUR wird nach der Bestimmung der Menge $V = \{x_1, x_2, \dots, x_k\}$ der k nächsten Nachbarn deren mittlerer Funk

⁵ Um das Beispiel einfach und anschaulich zu halten, wurde der Merkmalsvektor x bewusst eindimensional gehalten.

tionswert

$$\hat{f}(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^k f(\mathbf{x}_i) \quad (8.2)$$

berechnet und als Approximation \hat{f} für den Anfragevektor \mathbf{x} übernommen. Je größer k wird, desto glatter wird die Funktion \hat{f} .

8.3.2 Der Abstand ist relevant

Bei der praktischen Anwendung sowohl der diskreten als auch der stetigen Variante der k -Nearest-Neighbour-Methode tritt oft ein Problem auf. Bei größer werdendem k existieren typischerweise mehr Nachbarn mit großem Abstand als solche mit kleinem Abstand. Dadurch wird die Berechnung von \hat{f} durch weit entfernt liegende Nachbarn dominiert. Um dies zu verhindern, werden die k Nachbarn so gewichtet, dass weiter entfernte Nachbarn weniger Einfluss auf das Ergebnis haben. Bei dem Mehrheitsentscheid im k -NEARESTNEIGHBOUR Algorithmus erhalten die „Stimmen“ das Gewicht

$$w_i = \frac{1}{1 + \alpha d(\mathbf{x}, \mathbf{x}_i)^2}, \quad (8.3)$$

welches quadratisch mit dem Abstand abnimmt. Die Konstante α bestimmt, wie schnell die Gewichte mit dem Abstand abnehmen. Gleichung (8.2) wird nun ersetzt durch

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^k w_i f(\mathbf{x}_i)}{\sum_{i=1}^k w_i}.$$

Bei gleichverteilter Dichte der Punkte im Merkmalsraum stellen diese Formeln sicher, dass der Einfluss der Punkte für größer werdenden Abstand asymptotisch gegen null geht. Dadurch ist es dann problemlos möglich, alle Trainingsdaten zur Klassifikation beziehungsweise Approximation eines Vektors zu verwenden.

Um ein Gefühl für diese Methoden zu bekommen, ist in Abb. 8.19 die k -Nearest-Neighbour-Methode (in der oberen Zeile) verglichen mit der abstandsgewichteten Optimierung. Beide Methoden sind in der Lage zu generalisieren, bzw. zu mitteln oder Rauschen zu glätten, wenn die Zahl der Nachbarn der k -Nearest-Neighbour-Methode oder der Parameter α passend gesetzt wird. Die Diagramme zeigen schön, dass die abstandsgewichtete Methode eine viel glattere Approximation erzeugt als die k -Nearest-Neighbour-Methode. Bezüglich der Approximationsqualität kann diese sehr einfache Methode ohne weiteres mit aufwändigen Approximationsalgorithmen wie etwa nichtlineare Neuronale Netze, Support Vektor Maschinen oder Gauß'sche Prozesse konkurrieren.

Zu der in (8.3) angegebenen Gewichtungsfunktion (auch Kernel genannt) gibt es viele Alternativen. Zum Beispiel kann auch eine Gauß'sche oder andere Glockenkurve verwendet werden. In den meisten Anwendungen sind die Ergebnisse nicht sehr sensitiv auf die

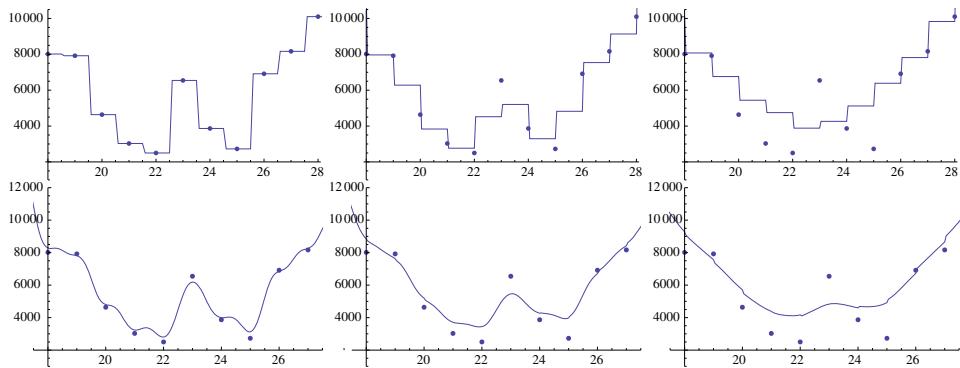


Abb. 8.19 Vergleich der k -Nearest-Neighbour-Methode (obere Reihe) mit $k = 1$ (links), $k = 2$ (Mitte) und $k = 6$ (rechts), mit der abstandsgewichteten Variante (untere Reihe) mit $\alpha = 20$ (links), $\alpha = 4$ (Mitte) und $\alpha = 1$ (rechts) auf einer eindimensionalen Datenmenge

genaue Wahl der Kernel-Funktionenklasse. Viel wichtiger ist die Wahl des Parameters α , was man in Abb. 8.19 sehr gut erkennt. Dieser Parameter kann manuell bestimmt werden. Es gibt aber auch verschiedene Methoden, diesen Parameter automatisch zu bestimmen [SA94, SE10]. In Abschn. 8.4.7 und 8.5 werden wir in Form der Kreuzvalidierung ein ganz allgemeines Verfahren zur Optimierung eines solchen Parameters kennenlernen.

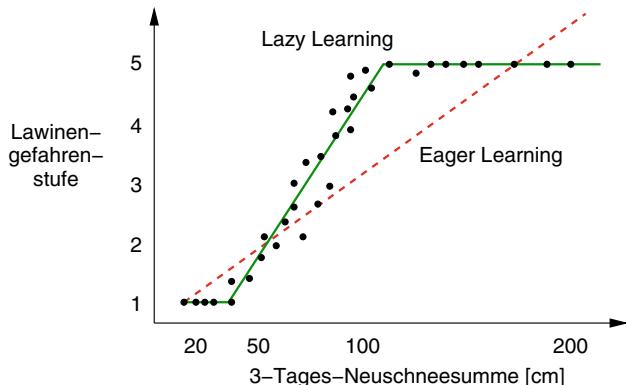
8.3.3 Rechenzeiten

Wie schon erwähnt, erfolgt das Lernen bei allen Varianten der Nearest Neighbour-Methode durch einfaches Abspeichern aller Trainingsvektoren zusammen mit deren Klassenwert, beziehungsweise dem Funktionswert $f(\mathbf{x})$. Daher gibt es kein anderes Lernverfahren, das so schnell lernt. Allerdings kann das Beantworten einer Anfrage zur Klassifikation oder Approximation eines Vektors \mathbf{x} sehr aufwändig werden. Allein das Finden der k nächsten Nachbarn bei n Trainingsdaten erfordert einen Aufwand, der linear mit n wächst. Für Klassifikation oder Approximation kommt dann noch ein linearer Aufwand in k dazu. Die Gesamtrechenzeit wächst also wie $\Theta(n + k)$. Bei großer Trainingsdatenanzahl kann dies zu Problemen führen.

8.3.4 Zusammenfassung und Ausblick

Weil bei den vorgestellten Nearest Neighbour-Methoden in der Lernphase außer dem Abspeichern der Daten nichts passiert, werden solche Verfahren auch als **Lazy Learning** (fauliges Lernen) bezeichnet, im Gegensatz zum **Eager Learning** (eifriges Lernen), bei dem die Lernphase aufwändig sein kann, aber die Anwendung auf neue Beispiele sehr effizient ist.

Abb. 8.20 Zur Bestimmung der Lawinengefahr wird aus Trainingsdaten eine Funktion approximiert. Hier im Vergleich ein lokales Modell (durchgezogene Linie) und ein globales (gestrichelt)



Das Perzeptron und alle anderen neuronalen Netze, das Lernen von Entscheidungsbäumen sowie das Lernen von Bayes-Netzen sind Methoden des Eager Learning. Da beim faulen Lernen die Trainingsdaten nur gespeichert werden und dieser Speicher dauerhaft für Klassifikation oder Approximation benötigt wird, wird das faule Lernen auch als Speicherbasiertes Lernen (engl. memory-based learning) bezeichnet.

Zum Vergleich zwischen diesen beiden Klassen von Lernverfahren wollen wir ein Beispiel verwenden. Wir stellen uns die Aufgabe, aus der Menge des in den letzten drei Tagen in einem bestimmten Gebiet der Schweiz gefallenen Neuschnees die aktuelle Lawinengefahr zu bestimmen.⁶ In Abb. 8.20 sind von Experten bestimmte Werte eingetragen, die wir als Trainingsdaten verwenden wollen. Bei der Anwendung eines fleißigen Lernverfahrens, das eine lineare Approximation der Daten vornimmt, wird die eingezeichnete gestrichelte Gerade errechnet. Aufgrund der Einschränkung auf eine Gerade ist der Fehler relativ groß. Er beträgt maximal etwa 1,5 Gefahrenstufen. Beim faulen Lernen wird erst beim Vorliegen des aktuellen Neuschneehöhenwertes die Gefahrenstufe bestimmt, und zwar aus einigen nächsten Nachbarn, das heißt lokal. Es könnte sich so die im Bild eingezeichnete Kurve ergeben, die sich aus Geradenstücken zusammensetzt und einen viel kleineren Fehler aufweist. Der Vorteil der faulen Methode ist die Lokalität. Die Approximation wird lokal um den aktuellen Neuschneehöhenwert vorgenommen und nicht global. Daher sind bei gleicher zu Grunde liegender Funktionenklasse (zum Beispiel Geraden) die faulen Verfahren besser.

Als Anwendung für Nearest Neighbour-Methoden eignen sich alle Problemstellungen, bei denen eine gute lokale Approximation benötigt wird, die aber keine hohen Anforderungen an die Geschwindigkeit des Systems stellen. Die hier erwähnte Lawinenprognose, die einmal täglich erfolgt, wäre eine solche Anwendung. Nicht geeignet sind die Nearest Neighbour-Methoden immer dann, wenn eine für Menschen verständliche Beschreibung

⁶ Die Dreitagesneuschneesumme ist zwar eines der wichtigsten Merkmale zur Bestimmung der Gefahrenstufe. In der Praxis werden aber noch andere Attribute verwendet [Bra01]. Das hier verwendete Beispiel ist vereinfacht.

Tab. 8.4 Einfaches Diagnosebeispiel mit einer Anfrage und dazu passendem Fall aus der Fallbasis

Merkmal	Anfrage	Fall aus Fallbasis
Defektes Teil	Rücklicht	Vorderlicht
Fahrrad Modell	Marin Pine Mountain	VSF T400
Baujahr	1993	2001
Stromquelle	Batterie	Dynamo
Zustand der Birnen	Ok	Ok
Lichtkabelzustand	?	Ok
Lösung		
Diagnose	?	Massekontakt vorne fehlt
Reparatur	?	Stelle Massekontakt vorne her

des in den Daten enthaltenen Wissens gefordert wird, wie dies heute bei vielen Data Mining-Anwendungen der Fall ist (siehe Abschn. 8.4). In den letzten Jahren werden diese speicherbasierten Methoden immer beliebter und verschiedene Verbesserungen wie zum Beispiel lokal gewichtete lineare Regression ((engl. locally weighted linear regression)) werden erfolgreich angewendet [Cle79].

Um die beschriebenen Methoden anwenden zu können, müssen die Trainingsdaten in Form von Vektoren aus ganzen oder reellen Zahlen vorliegen. Sie sind daher ungeeignet für Anwendungen, in denen die Daten symbolisch, zum Beispiel als prädikatenlogische Formel, vorliegen. Darauf gehen wir nun kurz ein.

8.3.5 Fallbasiertes Schließen

Beim fallbasierten Schließen (engl. case-based reasoning), kurz CBR, wird die Methode der Nächsten Nachbarn erweitert auf symbolische Problembeschreibungen und deren Lösung [Ric03]. CBR kommt zum Einsatz bei der Diagnose technischer Probleme im Kundendienst oder bei Telefonhotlines. Das in Tab. 8.4 dargestellte Beispiel zur Diagnose des Ausfalls eines Fahrradlichts verdeutlicht den Sachverhalt.

Für die Anfrage eines Kunden mit defektem Fahrradrücklicht wird eine Lösung gesucht. In der rechten Spalte ist ein zu der Anfrage in der mittleren Spalte ähnlicher Fall angegeben. Dieser entstammt der Fallbasis, welche bei der Nearest Neighbour-Methode den Trainingsdaten entspricht. Die Aufgabenstellung ist nun aber deutlich anspruchsvoller. Würde man wie bei der Nearest Neighbour-Methode einfach die Lösung des ähnlichsten Falles übernehmen, so würde bei dem defekten Rücklicht versucht, das Vorderlicht zu reparieren. Man benötigt also auch noch eine Rücktransformation der Lösung des gefundenen ähnlichen Falles auf die Anfrage. Die wichtigsten Schritte bei der Lösung eines CBR-Falles sind in Abb. 8.21 aufgeführt. Die Transformation in diesem Fall ist einfach: Rücklicht wird auf Vorderlicht abgebildet.

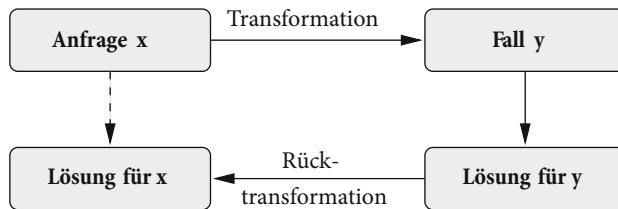


Abb. 8.21 Ist ein zum Fall x ähnlicher Fall y gefunden, so muss die Transformation bestimmt werden und diese dann invers auf den gefundenen Fall y angewendet werden, um eine Lösung für x zu erhalten

So schön und einfach diese Methode in der Theorie erscheint, in der Praxis ist der Bau von CBR-Diagnosesystemen ein sehr schwieriges Unterfangen. Die drei Hauptschwierigkeiten sind:

Modellierung Die Domäne der Anwendung muss in einem formalen Rahmen modelliert werden. Hierbei bereitet die aus Kap. 4 bekannte Monotonie der Logik Schwierigkeiten. Kann der Entwickler alle möglichen Spezialfälle und Problemvarianten vorhersehen und abbilden?

Ähnlichkeit Das Finden eines passenden Ähnlichkeitsmaßes für symbolische, nichtnumerische Merkmale.

Transformation Auch wenn ein ähnlicher Fall gefunden ist, ist es noch nicht klar, wie die Transformationsabbildung und deren Umkehrung auszusehen hat.

Es gibt zwar heute praktisch eingesetzte CBR-Systeme für Diagnoseanwendungen. Diese bleiben aber aus den genannten Gründen in ihrer Leistungsfähigkeit und Flexibilität noch weit hinter menschlichen Experten zurück. Eine interessante Alternative zu CBR sind die in Kap. 7 vorgestellten Bayes-Netze. Vielfach lässt sich die symbolische Problemrepräsentation auch ganz gut abbilden auf diskrete oder stetige numerische Merkmale. Dann sind die bewährten induktiven Lernverfahren wie etwa Entscheidungsbäume oder neuronale Netze erfolgreich einsetzbar.

8.4 Lernen von Entscheidungsbäumen

Das Entscheidungsbaumlernen ist ein für die praktische KI außerordentlich wichtiges, weil sehr mächtiges, gleichzeitig aber einfaches und effizientes Verfahren, um aus Daten Wissen zu extrahieren. Gegenüber den beiden bisher vorgestellten Verfahren besitzt es einen großen Vorteil. Das gewonnene Wissen ist nicht nur als Funktion (Blackbox, Programm) verfügbar und anwendbar, sondern kann in Form eines anschaulichen Entscheidungsbäumes von Menschen leicht verstanden, interpretiert und kontrolliert werden. Dies macht

Tab. 8.5 Variablen für das Klassifikationsproblem Skifahren

Variable	Werte	Beschreibung
<i>Skifahren</i> (Zielvariable)	ja, nein	Fahre ich los in das nächstgelegene Skigebiet mit ausreichend Schnee?
<i>Sonne</i> (Merkmal)	ja, nein	Sonne scheint heute?
<i>Schnee_Enf</i> (Merkmal)	$\leq 100, > 100$	Entfernung des nächsten Skigebiets mit guten Schneeverhältnissen (über/unter 100 km)
<i>Wochenende</i> (Merkmal)	ja, nein	Ist heute Wochenende?

die Induktion von Entscheidungsbäumen zu einem wichtigen Werkzeug auch für das Data Mining.

Wir werden die Funktion und auch die Anwendung des Entscheidungsbaumlernens anhand des Systems **C4.5** behandeln. C4.5 wurde 1993 von dem Australier Ross Quinlan vorgestellt und ist eine Weiterentwicklung seines Vorgängers **ID3** (Iterative Dichotomiser 3, 1986). Es ist für die nicht kommerzielle Anwendung frei verfügbar [Qui93]. Eine Weiterentwicklung, die noch effizienter arbeitet und Kosten von Entscheidungen berücksichtigen kann, ist C5.0 [Qui93].

Ganz ähnlich wie C4.5 arbeitet das System **CART** (Classification and Regression Trees, 1984) von Leo Breiman [BFOS84]. Es besitzt eine komfortable graphische Benutzeroberfläche, ist aber recht teuer.

Schon zwanzig Jahre früher wurde von J. Sonquist und J. Morgan mit dem System **CHAID** (Chi-square Automatic Interaction Detectors) 1964 das erste System vorgestellt, das automatisch Entscheidungsbäume erzeugen kann. Es hat die bemerkenswerte Eigenschaft, dass es das Wachsen des Baumes stoppt, bevor er zu groß wird, besitzt aber heute keine Bedeutung mehr.

Interessant ist auch das Data Mining-Werkzeug **KNIME** (Konstanz Information Miner), das eine komfortable Benutzeroberfläche bietet und unter Verwendung der **WEKA** Java-Bibliothek auch die Induktion von Entscheidungsbäumen ermöglicht. In Abschn. 8.10 werden wir KNIME vorstellen.

Zuerst werden wir nun an einem einfachen Beispiel zeigen, wie aus Trainingsdaten ein Entscheidungsbaum aufgebaut wird, um dann das Verfahren zu analysieren und auf ein komplexeres Beispiel zur medizinischen Diagnose anwenden.

8.4.1 Ein einfaches Beispiel

Für einen leidenschaftlichen Skifahrer, der in Alpennähe wohnt, soll ein Entscheidungsbaum aufgebaut werden, der ihm hilft, zu entscheiden, ob es sich lohnt, mit dem Auto in ein Skigebiet in den Bergen zu fahren. Wir haben also das Zweiklassenproblem **Skifahren ja/nein**, basierend auf den in Tab. 8.5 gelisteten Variablen.

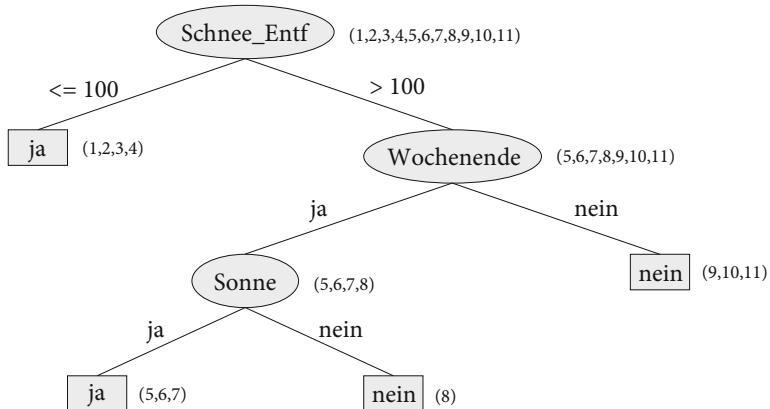


Abb. 8.22 Entscheidungsbaum für das Klassifikationsproblem Skifahren. In den Listen rechts neben den Knoten sind die Nummern der zugehörigen Trainingsdaten angegeben. Man beachte, dass der Blattknoten *Sonne = ja* nur zwei der drei Beispiele korrekt klassifiziert

Tab. 8.6 Datenmenge für das Klassifikationsproblem Skifahren

Tag	Schnee_Entf	Wochenende	Sonne	Skifahren
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

Abbildung 8.22 zeigt einen **Entscheidungsbaum** für dieses Problem. Ein Entscheidungsbaum ist ein Baum, dessen innere Knoten Merkmale (Attribute) repräsentieren. Jede Kante steht für einen Attributwert. An jedem Blattknoten ist ein Klassenwert angegeben.

Die zum Aufbau des Baumes im Beispiel verwendeten Daten sind in Tab. 8.6 eingetragen. Jede Zeile in der Tabelle enthält die Daten für einen Tag und stellt so ein Trainingsdatum (engl. sample) dar. Bei genauerer Betrachtung der Daten fällt auf, dass sich die Zeilen 6 und 7 widersprechen. Damit kann kein deterministischer Klassifikationsalgorithmus alle Daten korrekt klassifizieren. Die Anzahl falsch klassifizierter Daten muss daher ≥ 1 sein. Der Baum aus Abb. 8.22 klassifiziert die Daten also tatsächlich optimal.

Wie entsteht nun solch ein Baum aus den Daten? Zur Beantwortung dieser Frage beschränken wir uns zunächst auf diskrete Attribute mit endlich vielen Werten. Da auch die Zahl der Attribute endlich ist und pro Pfad jedes Attribut höchstens einmal vorkommen kann, gibt es endlich viele verschiedene Entscheidungsbäume. Ein einfacher, naheliegender Algorithmus zum Aufbau des Baumes würde einfach alle Bäume erzeugen und dann für jeden Baum die Zahl der Fehlklassifikationen auf den Daten berechnen und schließlich einen Baum mit minimaler Fehlerzahl auswählen. Damit hätten wir sogar einen optimalen Algorithmus (im Sinne des Fehlers auf den Trainingsdaten) zum Lernen von Entscheidungsbäumen.

Der offensichtliche Nachteil dieses Algorithmus ist seine inakzeptabel hohe Rechenzeit, sobald die Zahl der Attribute etwas größer wird. Wir entwickeln nun einen heuristischen Algorithmus, der, angefangen von der Wurzel, rekursiv einen Entscheidungsbaum aufbaut. Zuerst wird für den Wurzelknoten aus der Menge aller Attribute das Attribut mit dem höchsten Informationsgewinn (*Schnee_Entf*) gewählt. Für jeden Attributwert ($\leq 100, > 100$) entsteht ein Zweig im Baum. Nun wird für jeden Zweig dieser Prozess rekursiv wiederholt. Beim Erzeugen eines Knotens wird hierbei unter den noch nicht verwendeten Attributen im Sinne einer Greedy-Strategie immer das Attribut mit dem höchsten Informationsgewinn ausgewählt.

8.4.2 Die Entropie als Maß für den Informationsgehalt

Bei dem erwähnten Top-Down Verfahren zum Aufbau eines Entscheidungsbäumes soll jeweils das Attribut ausgewählt werden, welches den höchsten Informationsgewinn liefert. Wir beginnen mit der Entropie als einem Maß für den Informationsgehalt einer Trainingsdatenmenge D . Wenn wir an obigem Beispiel nur die binäre Variable *Skifahren* betrachten, so lässt sich D schreiben als

$$D = (\text{ja}, \text{ja}, \text{ja}, \text{ja}, \text{ja}, \text{nein}, \text{nein}, \text{nein}, \text{nein}, \text{nein})$$

mit geschätzten Wahrscheinlichkeiten

$$p_1 = P(\text{ja}) = 6/11 \quad \text{und} \quad p_2 = P(\text{nein}) = 5/11.$$

Wir haben hier offenbar eine Wahrscheinlichkeitsverteilung $\mathbf{p} = (6/11, 5/11)$. Allgemein für ein n Klassen Problem ergibt sich

$$\mathbf{p} = (p_1, \dots, p_n)$$

mit

$$\sum_{i=1}^n p_i = 1.$$

Zur Einführung des Informationsgehalts einer Verteilung betrachten wir zwei Extremfälle. Sei zuerst

$$\mathbf{p} = (1, 0, 0, \dots, 0). \quad (8.4)$$

Das heißt, das erste der n Ereignisse tritt sicher auf und alle anderen nie. Die Ungewissheit über den Ausgang des Ereignisses ist also minimal. Dagegen ist bei der Gleichverteilung

$$\mathbf{p} = \left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n} \right) \quad (8.5)$$

die Ungewissheit maximal, denn kein Ereignis hebt sich von den anderen ab. Claude Shannon hat sich nun gefragt, wie viele Bits man mindestens benötigt, um solch ein Ereignis zu kodieren. Im sicheren Fall von (8.4) werden null Bits benötigt, denn wir wissen, dass immer der Fall $i = 1$ eintritt. Im gleichverteilten Fall von (8.5) gibt es n gleich-wahrscheinliche Möglichkeiten. Bei binärer Kodierung werden hier $\log_2 n$ Bits benötigt. Da alle Einzelwahrscheinlichkeiten $p_i = 1/n$ sind, werden hier also $\log_2 \frac{1}{p_i}$ Bits zur Kodierung benötigt.

Im allgemeinen Fall $\mathbf{p} = (p_1, \dots, p_n)$, wenn die Wahrscheinlichkeiten der Elementareignisse von der Gleichverteilung abweichen, wird der Erwartungswert H für die Zahl der Bits berechnet. Dazu werden wir alle Werte $\log_2 \frac{1}{p_i} = -\log_2 p_i$ mit ihrer Wahrscheinlichkeit gewichten 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 man benötigt, um ein Ereignis zu kodieren, desto höher ist offenbar die Unsicherheit über den Ausgang. Daher definiert man:

Definition 8.4

Die **Entropie** H als Maß für die Unsicherheit einer Wahrscheinlichkeitsverteilung ist definiert durch⁷

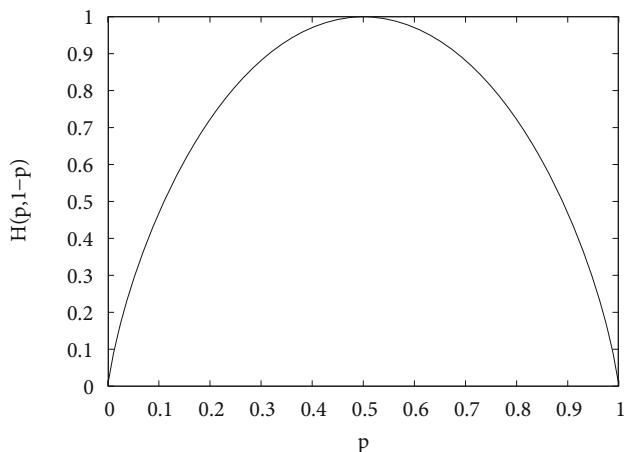
$$H(\mathbf{p}) = H(p_1, \dots, p_n) := -\sum_{i=1}^n p_i \log_2 p_i.$$

Eine detaillierte Herleitung dieser Formel ist in [SW76] zu finden. Setzt man hier das sichere Ereignis $\mathbf{p} = (1, 0, 0, \dots, 0)$ ein, so ergibt sich durch $0 \log_2 0$ ein undefinierter Ausdruck. Dieses Problem löst man durch die Definition $0 \log_2 0 := 0$ (siehe Aufgabe 8.10).

⁷ In (7.9) wird zur Definition der Entropie der natürliche Logarithmus verwendet. Da hier und auch bei der MaxEnt-Methode nur Entropien verglichen werden, spielt dieser Unterschied keine Rolle (siehe Aufgabe 8.12).

Abb. 8.23 Die Entropiefunktion für den Zweiklassenfall.

Man erkennt das Maximum bei $p = 1/2$ und die Symmetrie bezüglich Vertauschung von p und $1 - p$



Nun können wir also $H(1, 0, \dots, 0) = 0$ berechnen. Wir werden zeigen, dass die Entropie im Hyperkubus $[0, 1]^n$ unter der Nebenbedingung $\sum_{i=1}^n p_i = 1$ bei der Gleichverteilung $(\frac{1}{n}, \dots, \frac{1}{n})$ ihren maximalen Wert annimmt. Im Fall eines Ereignisses mit zwei möglichen Ausgängen, welcher der Klassifikation mit zwei Klassen entspricht, ergibt sich

$$H(\mathbf{p}) = H(p_1, p_2) = H(p_1, 1 - p_1) = -(p_1 \log_2 p_1 + (1 - p_1) \log_2 (1 - p_1)).$$

Dieser Ausdruck ist in Abb. 8.23 in Abhängigkeit von p_1 dargestellt. Das Maximum liegt bei $p_1 = 1/2$.

Da jeder klassifizierten Datenmenge D mittels Schätzen der Klassenwahrscheinlichkeiten eine Wahrscheinlichkeitsverteilung \mathbf{p} zugeordnet ist, können wir den Entropiebegriff auf Daten erweitern durch die Definition

$$H(D) = H(\mathbf{p}).$$

Unter dem Informationsgehalt $I(D)$ der Datenmenge D versteht man das Gegenteil von Unsicherheit. Also definieren wir:

Definition 8.5

Der Informationsgehalt einer Datenmenge ist definiert als

$$I(D) := 1 - H(D). \quad (8.6)$$

8.4.3 Der Informationsgewinn

Wenden wir die Entropieformel an auf das Beispiel, so ergibt sich

$$H(6/11, 5/11) = 0,994$$

Beim Aufbau eines Entscheidungsbaumes wird die Datenmenge durch jedes neue Attribut weiter unterteilt. Ein Attribut ist umso besser, je mehr seine Aufteilung den Informationsgehalt der Verteilung erhöht. Entsprechend definieren wir:

Definition 8.6

Der **Informationsgewinn** $G(D, A)$ durch die Verwendung des Attributs A wird bestimmt durch die Differenz aus dem mittleren Informationsgehalt der durch das n -wertige Attribut A aufgeteilten Datenmenge $D = D_1 \cup D_2 \cup \dots \cup D_n$ und dem Informationsgehalt $I(D)$, was

$$G(D, A) = \sum_{i=1}^n \frac{|D_i|}{|D|} I(D_i) - I(D)$$

ergibt.

Mit (8.6) erhalten wir daraus

$$\begin{aligned} G(D, A) &= \sum_{i=1}^n \frac{|D_i|}{|D|} I(D_i) - I(D) = \sum_{i=1}^n \frac{|D_i|}{|D|} (1 - H(D_i)) - (1 - H(D)) \\ &= 1 - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) - 1 + H(D) \\ &= H(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i). \end{aligned} \tag{8.7}$$

Angewendet auf unser Beispiel für das Attribut *Schnee_Entf* ergibt sich

$$\begin{aligned} G(D, Schnee_Entf) &= H(D) - \left(\frac{4}{11} H(D_{\leq 100}) + \frac{7}{11} H(D_{> 100}) \right) \\ &= 0,994 - \left(\frac{4}{11} \cdot 0 + \frac{7}{11} \cdot 0,863 \right) = 0,445. \end{aligned}$$

Analog erhält man

$$G(D, Wochenede) = 0,150$$

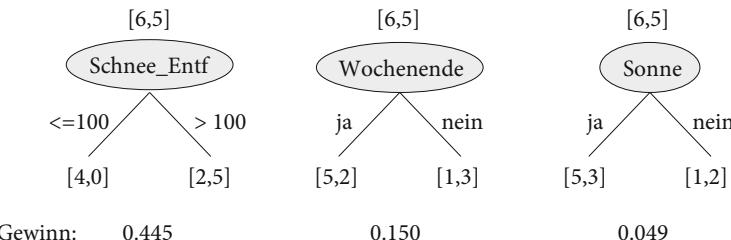


Abb. 8.24 Der für die verschiedenen Attribute berechnete Gewinn spiegelt wider, ob die Aufteilung der Daten durch das jeweilige Attribut zu einer besseren Klassentrennung führt. Je deutlicher die durch die Attribute erzeugten Verteilungen der Daten von der Gleichverteilung abweichen, desto höher ist der Informationsgewinn

und

$$G(D, \text{Sonne}) = 0,049.$$

Das Attribut *Schnee_Entf* wird nun also zum Wurzelknoten für den Entscheidungsbaum. Die Situation bei der Auswahl dieses Attributs ist in Abb. 8.24 nochmal 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 Klassifikation eindeutig *ja*. Also terminiert der Baum hier. Im anderen Zweig $D_{> 100}$ herrscht keine Eindeutigkeit. Also wird hier rekursiv das Verfahren wiederholt. Aus den beiden noch verfügbaren Attributen *Sonne* und *Wochenende* muss das beste ausgewählt werden. Wir berechnen

$$G(D_{> 100}, \text{Wochenende}) = 0,292$$

und

$$G(D_{> 100}, \text{Sonne}) = 0,170.$$

Der Knoten erhält also das Attribut *Wochenende*. Für *Wochenende* = *nein* terminiert der Baum mit der Entscheidung *nein*. Eine Berechnung des Gewinns liefert hier den Wert 0. Für *Wochenende* = *ja* bringt *Sonne* noch einen Gewinn von 0,171. Dann terminiert der Aufbau des Baumes, weil keine weiteren Attribute mehr verfügbar sind, obwohl das Beispiel Nummer 7 falsch klassifiziert wird. Der fertige Baum ist aus Abb. 8.22 schon bekannt.

8.4.4 Die Anwendung von C4.5

Den soeben erzeugten Entscheidungsbäum können wir auch von C4.5 erzeugen lassen. Die Trainingsdaten werden in einer Datei *ski.data* folgendermaßen gespeichert:

```

<=100, ja , ja , ja
<=100, ja , 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 über Attribute und Klassen werden in der Datei `ski.names` abgelegt (Zeilen mit | am Anfang sind Kommentarzeilen):

```

|Klassen:      nein: nicht Skifahren, ja: Skifahren
|
nein,ja.
|
|Attribute
|
Schnee_Entf:          <=100,>100.
Wochenende:           nein,ja.
Sonne:                nein,ja.

```

In der Unix-Kommandozeile wird C4.5 dann aufgerufen und erzeugt den unten dargestellten, durch Einrückungen formatierten Entscheidungsbaum. Die Option `-f` dient der Angabe des Dateinamens (ohne Extension), und über die Option `-m` wird die minimale Zahl von Trainingsdaten für das Erzeugen eines neuen Astes im Baum angegeben. Weil die Zahl der Trainingsdaten in diesem Beispiel extrem klein war, ist `-m 1` hier sinnvoll. Bei größerer Datenmenge sollte mindestens `-m 10` verwendet werden.

```

unixprompt> c4.5 -f ski -m 1
C4.5 [release 8] decision tree generator      Wed Aug 23 10:44:49 2006
-----
Options:
  File stem <ski>
  Sensible test requires 2 branches with >=1 cases

Read 11 cases (3 attributes) from ski.data

```

Decision Tree:

```

Schnee_Entf = <=100: ja (4.0)
Schnee_Entf = >100:
|   Wochenende = nein: nein (3.0)
|   Wochenende = ja:
|   |   Sonne = nein: nein (1.0)
|   |   Sonne = ja: ja (3.0/1.0)

```

Simplified Decision Tree:

```

Schnee_Entf = <=100: ja (4.0/1.2)
Schnee_Entf = >100: nein (7.0/3.4)

```

Evaluation on training data (11 items):

Before Pruning		After Pruning		
Size	Errors	Size	Errors	Estimate
7	1 (9.1%)	3	2 (18.2%)	(41.7%) <<

Es ist außerdem ein vereinfachter Baum mit nur einem Attribut angegeben. Dieser durch Pruning (siehe Abschn. 8.4.7) entstandene Baum ist wichtig bei größer werdender Zahl von Trainingsdaten. Bei diesem kleinen Beispiel macht er noch nicht viel Sinn. Auch die Fehlerrate für beide Bäume auf den Trainingsdaten wird angegeben. Die Zahlen in Klammern hinter den Entscheidungen geben die Größe der zugrunde liegenden Datenmenge und die Zahl der Fehler an. Zum Beispiel die Zeile Sonne = ja: ja (3.0/1.0) in obigem Baum besagt, dass für diesen Blattknoten Sonne = ja drei Trainingsbeispiele existieren, von denen eines falsch klassifiziert wird. Der Benutzer kann hier also ablesen, ob die Entscheidung statistisch fundiert und/oder sicher ist.

Wir können nun in Abb. 8.25 das Schema des Lernalgorithmus zum Erzeugen eines Entscheidungsbaumes angeben.

Die Grundlagen des automatischen Erzeugens von Entscheidungsbäumen sind nun bekannt. Für den Einsatz in der Praxis werden aber wichtige Erweiterungen benötigt. Diese wollen wir anhand der schon bekannten LEXMED-Anwendung einführen.

8.4.5 Lernen von Appendizitisdiagnose

Im Forschungsprojekt LEXMED wurde, basierend auf einer Datenbank mit Patientendaten, ein Expertensystem zur Diagnose der Blinddarmentzündung entwickelt [ES99, SE00]. Das mit der Methode der maximalen Entropie arbeitende System ist in Abschn. 7.3 beschrieben.

ERZEUGEENTSCHEIDUNGSBAUM(*Daten, Knoten*)

A_{max} = Attribut mit maximalem Informationsgewinn

If $G(A_{max}) = 0$

Then *Knoten* wird Blattknoten mit häufigster Klasse in *Daten*

Else ordne *Knoten* das Attribut A_{max} zu

Erzeuge für jeden Wert a_1, \dots, a_n von A_{max}

einen Nachfolgeknoten: K_1, \dots, K_n

Teile *Daten* auf in D_1, \dots, D_n mit $D_i = \{x \in \text{Daten} | A_{max}(x) = a_i\}$

For all $i \in \{1, \dots, n\}$

If alle $x \in D_i$ gehören zur gleichen Klasse C_i

Then Erzeuge Blattknoten K_i mit Klasse C_i

Else ERZEUGEENTSCHEIDUNGSBAUM(D_i, K_i)

Abb. 8.25 Algorithmus für den Aufbau eines Entscheidungsbaumes

Wir verwenden die LEXMED-Datenbank, um mit C4.5 einen Entscheidungsbaum zur Diagnose der Blinddarmentzündung zu erzeugen. Die als Attribute verwendeten Symptome sind in der Datei app.names definiert:

```
|Definition der Klassen und Attribute
|
|Klassen      0=Appendizitis negativ
|                  1=Appendizitis positiv
0,1.
|
|Attribute
|
|Alter:                      continuous.
Geschlecht_(1=m___2=w):      1,2.
Schmerz_Quadrant1_(0=nein__1=ja):    0,1.
Schmerz_Quadrant2_(0=nein__1=ja):    0,1.
Schmerz_Quadrant3_(0=nein__1=ja):    0,1.
Schmerz_Quadrant4_(0=nein__1=ja):    0,1.
Lokale_Abwehrspannung_(0=nein__1=ja): 0,1.
Generalisierte_Abwehrspannung_(0=nein__1=ja): 0,1.
Schmerz_bei_Loslassmanoever_(0=nein__1=ja): 0,1.
Erschuetterung_(0=nein__1=ja):        0,1.
Schmerz_bei_rektaler_Untersuchung_(0=nein__1=ja): 0,1.
Temp_ax:                         continuous.
Temp_re:                          continuous.
Leukozyten:                      continuous.
Diabetes_mellitus_(0=nein__1=ja):   0,1
```

Man erkennt, dass neben vielen binären Attributen wie zum Beispiel die verschiedenen Schmerz-Symptome auch stetige Symptome wie das Alter oder die Fiebertemperatur vor-

kommen. In der folgenden Datei `app.data` mit den Trainingsdaten ist in jeder Zeile ein Fall beschrieben. In der ersten Zeile etwa ein 19jähriger männlicher Patient mit Schmerz im dritten Quadranten (rechts unten, wo der Blinddarm sitzt), den beiden Fieberwerten 36,2 und 37,8 Grad, einem Leukozytenwert von 13.400 und positivem Befund, das heißt entzündetem Blinddarm.

```
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  
...
```

Ohne im Detail auf die Datenbank einzugehen, ist es wichtig, zu erwähnen, dass in der Datenbank nur Patienten erfasst sind, die mit Verdacht auf Appendizitis im Krankenhaus erschienen sind und dann operiert wurden. Man erkennt in der siebten Zeile, dass C4.5 auch mit fehlenden Werten umgehen kann. Die Datei enthält 9764 Fälle.

```
unixprompt> c4.5 -f app -u -m 100
```

```
C4.5 [release 8] decision tree generator           Wed Aug 23 13:13:15 2006
```

```
-----  
Read 9764 cases (15 attributes) from app.data
```

```
Decision Tree:
```

```
Leukozyten <= 11030 :  
|   Schmerz_bei_Loslassmanoever = 0:  
|   |   Temp_re > 381 : 1 (135.9/54.2)  
|   |   Temp_re <= 381 :  
|   |   |   Lokale_Abwehrspannung = 0: 0 (1453.3/358.9)  
|   |   |   Lokale_Abwehrspannung = 1:  
|   |   |   |   Geschlecht_(1=m__2=w) = 1: 1 (160.1/74.9)  
|   |   |   |   Geschlecht_(1=m__2=w) = 2: 0 (286.3/97.6)  
|   Schmerz_bei_Loslassmanoever = 1:  
|   |   Leukozyten <= 8600 :  
|   |   |   Temp_re > 378 : 1 (176.0/59.4)
```

```

|   |   |   Temp_re <= 378 :
|   |   |   |   Geschlecht_(1=m___2=w) = 1:
|   |   |   |   |   Lokale_Abwehrspannung = 0: 0 (110.7/51.7)
|   |   |   |   |   Lokale_Abwehrspannung = 1: 1 (160.6/68.5)
|   |   |   |   Geschlecht_(1=m___2=w) = 2:
|   |   |   |   |   Alter <= 14 : 1 (131.1/63.1)
|   |   |   |   |   Alter > 14 : 0 (398.3/137.6)
|   |   Leukozyten > 8600 :
|   |   |   |   Geschlecht_(1=m___2=w) = 1: 1 (429.9/91.0)
|   |   |   |   Geschlecht_(1=m___2=w) = 2:
|   |   |   |   |   Lokale_Abwehrspannung = 1: 1 (311.2/103.0)
|   |   |   |   |   Lokale_Abwehrspannung = 0:
|   |   |   |   |   Lokale_Abwehrspannung = 1:
|   |   |   |   |   Geschlecht_(1=m___2=w) = 1: 1 (160.1/79.7)
|   |   |   |   |   Geschlecht_(1=m___2=w) = 2: 0 (286.3/103.7)
|   |   Schmerz_bei_Loslassmanoever = 1:
|   |   |   Leukozyten > 8600 : 1 (984.7/322.6)
|   |   Leukozyten <= 8600 :
|   |   |   Temp_re > 378 : 1 (176.0/64.3)
|   |   |   Temp_re <= 378 :
|   |   |   |   Geschlecht_(1=m___2=w) = 1:
|   |   |   |   |   Lokale_Abwehrspannung = 0: 0 (110.7/55.8)
|   |   |   |   |   Lokale_Abwehrspannung = 1: 1 (160.6/73.4)
|   |   |   |   Geschlecht_(1=m___2=w) = 2:
|   |   |   |   |   Alter <= 14 : 1 (131.1/67.6)
|   |   |   |   |   Alter > 14 : 0 (398.3/144.7)

```

Evaluation on training data (9764 items):

Before Pruning		After Pruning		
Size	Errors	Size	Errors	Estimate
37	2197(22.5%)	21	2223(22.8%)	(23.6%) <<

Evaluation on test data (4882 items):

Before Pruning		After Pruning		
Size	Errors	Size	Errors	Estimate
37	1148(23.5%)	21	1153(23.6%)	(23.6%) <<
(a) (b)		<-classified as		
758	885	(a): class 0		
268	2971	(b): class 1		

8.4.6 Stetige Attribute

In den erzeugten Bäumen für die Appendizitisdiagnose gibt es einen Knoten Leukozyten > 11030 , der offenbar entstanden ist aus dem stetigen Attribut Leukozyten durch das Setzen einer Schwelle auf den Wert 11.030. C4.5 hat also aus dem stetigen Attribut Leukozyten ein binäres Attribut Leukozyten > 11030 gemacht. Die Schwelle $\Theta_{D,A}$ für ein Attribut A wird bestimmt durch folgendes Verfahren: Für alle in den Trainingsdaten D vorkommenden Werte v von A wird das binäre Attribut $A > v$ erzeugt und hierfür der Informationsgewinn berechnet. Die Schwelle $\Theta_{D,A}$ wird dann auf den Wert v mit dem maximalen Informationsgewinn gesetzt, das heißt es gilt

$$\Theta_{D,A} = \underset{v}{\operatorname{argmax}} \{ G(D, A > v) \}.$$

Bei einem Attribut wie dem Leukozytenwert oder auch dem Alter ist eine Entscheidung basierend auf einer zweiseitigen Diskretisierung vermutlich zu ungenau. Trotzdem besteht keine Notwendigkeit, feiner zu diskretisieren, denn jedes stetige Attribut wird an jedem neu erzeugten Knoten wieder getestet und kann so mit dem am jeweiligen Knoten optimalen Wert wiederholt vorkommen. Damit erhält man letztendlich eine sehr gute Diskretisierung, deren Feinheit dem Problem angemessen ist.

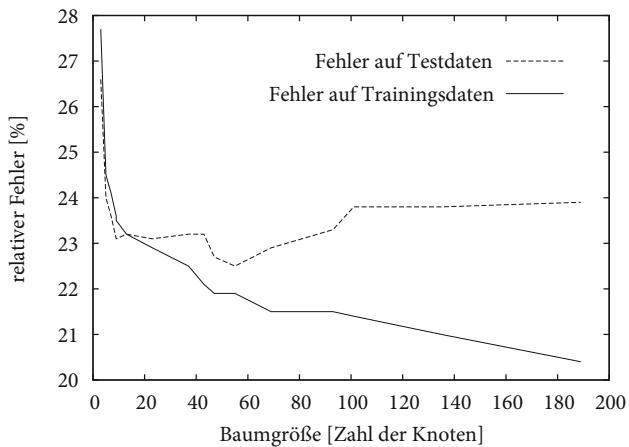
8.4.7 Pruning – Der Baumschnitt

Schon zur Zeit von Aristoteles wurde gefordert, dass von zwei wissenschaftlichen Theorien, die den gleichen Sachverhalt gleich gut erklären, die einfachere zu bevorzugen ist. Dieses Sparsamkeitsprinzip, das heute bekannt ist unter dem Namen **Ockhams Rasiermesser** (engl. Occam's razor), ist auch beim maschinellen Lernen und Data Mining von großer Wichtigkeit.

Ein Entscheidungsbaum ist nichts anderes als eine Theorie zur Beschreibung der Trainingsdaten. Eine andere Theorie zur Beschreibung der Daten sind die Daten selbst. Wenn nun der Baum alle Daten ohne Fehler klassifiziert, aber viel kleiner und daher für uns Menschen einfacher zu verstehen ist, so ist er nach Ockhams Rasiermesser zu bevorzugen. Das gleiche gilt für zwei verschieden große Entscheidungsbäume. Auch hier ist der kleinere, bzw. der einfachere zu bevorzugen. Also muss es das Ziel eines jeden Verfahrens zum Erzeugen eines Entscheidungsbaumes sein, bei gegebener Fehlerrate einen möglichst kleinen Entscheidungsbaum zu erzeugen. Unter allen Bäumen mit einer festen Fehlerrate ist also immer ein kleinster Baum auszuwählen.

Bisher wurde der Begriff der Fehlerrate noch nicht genauer definiert. Wie schon mehrfach erwähnt, ist es wichtig, dass der gelernte Baum nicht nur die Trainingsdaten gut auswendig lernt, sondern gut generalisiert. Um die Generalisierungsfähigkeit eines Baumes zu testen, teilen wir die verfügbaren Daten auf in eine Menge von **Trainingsdaten** und eine Menge von **Testdaten**. Die Testdaten werden dem Lernalgorithmus vorenthalten und

Abb. 8.26 Lernkurve von C4.5 auf den Appendizitisdaten. Man erkennt deutlich die Überanpassung bei Bäumen mit mehr als 55 Knoten



nur zum Testen verwendet. Sind viele Daten verfügbar, wie hier bei den Appendizitisdaten, so kann man zum Beispiel 2/3 der Daten zum Lernen und das andere Drittel zum Testen verwenden.

Das Ockhamsche Rasiermesser hat neben der besseren Verständlichkeit noch einen anderen wichtigen Grund, nämlich die Generalisierungsfähigkeit. Je komplexer ein Modell (hier ein Entscheidungsbaum), desto mehr Details werden repräsentiert, aber umso schlechter lässt sich das Modell auf neue Daten übertragen. Dieser Zusammenhang ist in Abb. 8.26 gut erkennbar. Auf den Appendizitisdaten wurden Entscheidungsbäume unterschiedlicher Größe trainiert. In der Graphik ist sowohl der Klassifikationsfehler auf den Trainingsdaten als auch der Fehler auf den (beim Lernen unbekannten) Testdaten eingezeichnet. Der Fehler auf den Trainingsdaten nimmt monoton ab mit der Größe des Baumes. Bis zu einer Größe von 55 Knoten nimmt auch der Fehler auf den Testdaten ab. Wächst der Baum weiter, dann nimmt der Fehler wieder zu! Diesen schon von der Nearest Neighbour-Methode bekannten Effekt nennt man **Überanpassung**, (engl. **overfitting**).

Diesen für fast alle Lernverfahren sehr wichtigen Begriff wollen wir in Anlehnung an [Mit97] allgemein definieren:

Definition 8.7

Sei ein bestimmtes Lernverfahren, das heißt eine Klasse lernender Agenten, gegeben. Man nennt einen Agenten A überangepasst an die Trainingsdaten, wenn es einen anderen Agenten A' gibt, dessen Fehler auf den Trainingsdaten größer ist als der von A , aber auf der gesamten Verteilung von Daten ist der Fehler von A' kleiner als der von A .

Wie kann man nun diesen Punkt des Fehlerminimums auf den Testdaten finden? Das naheliegendste Verfahren nennt sich **Kreuzvalidierung (engl. cross validation)**. Dabei wird während des Aufbaus des Baumes mit den Trainingsdaten parallel immer der Fehler auf den Testdaten gemessen. Sobald der Fehler signifikant ansteigt, wird der Baum mit dem minimalen Fehler gespeichert. Dieses Verfahren wird bei dem erwähnten System CART angewendet.

C4.5 arbeitet hier etwas anders. Es erzeugt zuerst mit dem Algorithmus ERZEUGE-ENTSCHEIDUNGSBAUM aus Abb. 8.25 einen meist überangepassten Baum. Dann wird mittels des so genannten **Pruning** solange versucht, beliebige Knoten des Baumes abzuschneiden, bis ein aus dem Fehler auf den Trainingsdaten geschätzter Fehler auf den Testdaten wieder zunimmt.⁸ Dies ist wie der Aufbau des Baumes auch ein Greedy-Verfahren. Das heißt, wenn einmal ein Knoten abgeschnitten ist, kann er nicht mehr hinzugefügt werden, auch wenn sich dies später als besser herausstellen sollte.

8.4.8 Fehlende Werte

Häufig fehlen in manchen Trainingsdaten einzelne Attributwerte. In den LEXMED-Daten etwa kommt folgender Eintrag vor

56, 1, 1, 1, 1, 0, 1, 1, 1, 0, 390, ?, 14.100, 0, 1,

bei dem einer der beiden Fieber-Wert fehlt. Solche Daten können beim Aufbau des Entscheidungsbaumes trotzdem verwendet werden. Man kann dem Attribut den häufigsten Wert unter allen anderen Trainingsdaten zuweisen oder den häufigsten Wert unter allen anderen Trainingsdaten in der gleichen Klasse. Noch besser ist es, für das fehlende Attribut die Wahrscheinlichkeitsverteilung aller Attributwerte einzusetzen und das fehlerhafte Trainingsbeispiel entsprechend dieser Verteilung aufzuteilen auf die verschiedenen Äste im Baum. Dies ist übrigens ein Grund für das Vorkommen von nicht ganzzahligen Werten in den Klammerausdrücken hinter den Blattknoten der C4.5-Bäume.

Nicht nur beim Lernen, sondern auch bei der Klassifikation können fehlende Werte auftreten. Diese werden gleich behandelt wie beim Lernen.

8.4.9 Zusammenfassung

Das Lernen von Entscheidungsbäumen gehört für Klassifikationsaufgaben zu den beliebtesten Verfahren. Gründe hierfür sind die einfache Anwendung und die Geschwindigkeit. Auf den etwa 10.000 LEXMED-Daten mit 15 Attributen benötigt C4.5 auf einem 3 Giga-

⁸ Besser wäre es allerdings, beim Pruning den Fehler auf den Testdaten zu verwenden. Zumindest dann, wenn die Zahl der Trainingsdaten ausreicht, um eine separate Testmenge zu rechtfertigen.

hertz-Rechner etwa 0,3 Sekunden für das Lernen. Dies ist im Vergleich zu anderen Lernverfahren sehr schnell.

Wichtig ist für den Anwender aber auch, dass er den Entscheidungsbaum als gelerntes Modell verstehen und gegebenenfalls abändern kann. Auch ist es nicht schwierig, gegebenenfalls auch automatisch einen Entscheidungsbaum in eine If-Then-Else-Kaskade zu überführen und damit sehr effizient in bestehende Programme einzubauen.

Da sowohl beim Aufbau des Baumes als auch beim Pruning ein Greedy-Verfahren verwendet wird, sind die Bäume im allgemeinen nicht optimal. Der gefundene Entscheidungsbaum hat zwar meist eine relativ kleine Fehlerrate, aber es gibt eventuell noch einen besseren Baum. C4.5 bevorzugt eben kleine Bäume und Attribute mit hohem Informationsgewinn ganz oben im Baum. Bei Attributen mit vielen Werten zeigt die vorgestellte Formel für den Informationsgewinn Schwächen. In [Mit97] sind Alternativen hierzu angegeben.

8.5 Kreuzvalidierung und Überanpassung

Wie in Abschn. 8.4.7 angesprochen, haben viele Lernverfahren das Problem der Überanpassung. Bei guten Lernverfahren, wie etwa beim Entscheidungsbaumlernen kann sich die Komplexität des gelernten Modells der Komplexität der Trainingsdaten anpassen. Dies führt zu Überanpassung, wenn in den Daten Rauschen vorkommt.

Bei der Kreuzvalidierung (engl. cross validation) versucht man, die Modellkomplexität so zu optimieren, dass der Klassifikations- oder Approximationsfehler auf einer beim Lernen unbekannten Testdatenmenge minimal wird. Dazu muss die Modellkomplexität über einen Parameter γ steuerbar sein. Beim Entscheidungsbaumlernen ist dies zum Beispiel die Baumgröße. Bei der k -Nearest-Neighbour-Methode aus Abschn. 8.3 ist dies die Zahl k der nächsten Nachbarn oder bei neuronalen Netzen die Zahl der verdeckten Neuronen (siehe Kap. 9).

Wir variieren nun also einen Parameter γ und testen das auf der Trainingsdatenmenge trainierte Verfahren auf einer unabhängigen Testdatenmenge, um dann den Wert von γ zu wählen, der den Fehler auf der Testdatenmenge minimiert. Die k -fache Kreuzvalidierung arbeitet nach folgendem Schema:

KREUZVALIDIERUNG(\mathbf{X}, k)

Daten zufällig aufteilen in k gleich große Blöcke $\mathbf{X} = \mathbf{X}_1 \cup \dots \cup \mathbf{X}_k$

For all $\gamma \in \{\gamma_{min}, \dots, \gamma_{max}\}$

For all $i \in \{1, \dots, k\}$

Trainiere ein Modell der Komplexität γ auf $\mathbf{X} \setminus \mathbf{X}_i$

Berechne den Fehler $E(\gamma, \mathbf{X}_i)$ auf der Testdatenmenge \mathbf{X}_i

Berechne den mittleren Fehler $E(\gamma) = \frac{1}{k} \sum_{i=1}^k E(\gamma, \mathbf{X}_i)$

Wähle den Wert $\gamma_{opt} = \operatorname{argmin}_{\gamma} E(\gamma)$ mit geringstem mittlerem Fehler

Trainiere finales Modell mit γ_{opt} auf gesamter Datenmenge \mathbf{X}

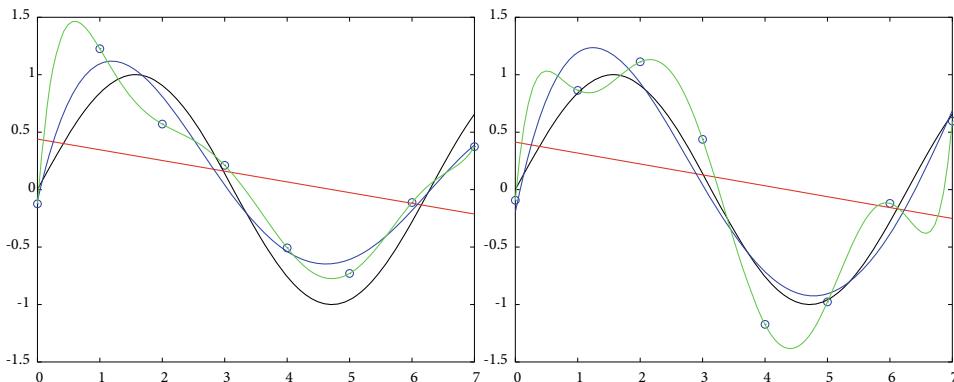


Abb. 8.27 Auf zwei unterschiedlichen Datenmengen wurden Polynome der Grade 1, 4 und 7 approximiert. Man erkennt gut den Bias-Variance-Tradeoff

Die gesamte verfügbare Datenmenge X wird in k gleich große Blöcke zerlegt. Dann wird k -mal der Algorithmus auf den Daten aus $k-1$ Blöcken trainiert und auf dem verbleibenden Block getestet. Über die k ermittelten Fehler wird gemittelt und dann der Wert γ_{opt} mit dem kleinsten mittleren Fehler zum Training des finalen Modells auf allen Daten X gewählt.

Wenn die Trainingsdatenmenge X groß ist, wird sie zum Beispiel in $k = 3$ oder $k = 10$ Blöcke zerlegt. Der dadurch entstehende 4-fache beziehungsweise 11-fache Rechenaufwand ist meist noch akzeptabel. Bei kleiner Trainingsdatenmenge will man mit möglichst allen n Merkmalvektoren trainieren. Dann kann man auch $k = n$ wählen und erhält die sogenannte Leave-One-Out-Kreuzvalidierung.

Die Kreuzvalidierung ist das wichtigste und am meisten verwendete Verfahren zur automatischen Optimierung der Modellkomplexität. Sie löst das Problem der Überanpassung. Eine andere Sicht auf dieses Problem erhält man aus dem Blickwinkel des sogenannten *Bias-Variance-Tradeoff*. Wenn ein zu einfaches Modell verwendet wird, erzwingt dieses eine nicht optimale Approximation der Daten in eine bestimmte Richtung (Bias). Wird hingegen ein zu komplexes Modell verwendet, so ist dieses in der Lage, sich an beliebig komplexe Datenmengen überanzupassen. Bei einer neuen Datenstichprobe aus der gleichen Verteilung kann so ein stark unterschiedliches Modell gelernt werden. Das Modell variiert also sehr bei veränderten Daten (Variance).

Schön veranschaulichen lässt sich dieser Zusammenhang am Beispiel der Funktionsapproximation mit Polynomen. Wählt man als Parameter γ den Grad des Polynoms, so wird beim Grad 1 eine Gerade approximiert und man erhält bei nicht trivialen Daten keine gute Approximation. Wenn hingegen der Grad gleich $n-1$ ist bei n Punkten, ist das Polynom in der Lage, alle Punkte zu treffen und den Fehler auf den Trainingsdaten auf null zu reduzieren. Dies kann man gut erkennen in Abb. 8.27 links. Hier wurden acht Datenpunkte nach der Formel

$$y(x) = \sin(x) + r(0, 0, 2)$$

erzeugt, wobei $r(0, 0, 2)$ ein normalverteiltes Rauschen mit Mittelwert 0 und Standardabweichung 0,2 erzeugt.

Im linken Bild ist neben den Punkten die zu Grunde liegende Sinusfunktion (schwarz) eingetragen sowie eine mit der Methode der kleinsten Quadrate (siehe Abschn. 9.4 oder [Ert12]) approximierte Gerade (rot). Außerdem wurde ein Polynom vom Grad sieben (grün) durch die Punkte interpoliert, das alle Punkte trifft, sowie ein Polynom vom Grad vier (blau), welches der Sinuskurve am nächsten kommt. Im rechten Bild wurden mit acht neuen nach obiger Formel generierten Punkten die gleichen Approximationen nochmal durchgeführt. Man erkennt sehr schön, dass trotz verschiedener Daten beide Geraden zwar stark von den Daten abweichen (Bias groß), aber einen sehr ähnlichen Verlauf haben (Variance sehr klein). Die beiden Polynome siebten Grades hingegen approximieren die Datenpunkte perfekt (Bias gleich null), aber sie haben einen sehr unterschiedlichen Verlauf (Variance sehr groß). Wir haben hier einen ganz klaren Überanpassungseffekt. Einen guten Kompromiss stellen die Polynome vierten Grade dar. Die Kreuzvalidierung würde vermutlich hier den Grad vier als optimale Lösung liefern.

8.6 Lernen von Bayes-Netzen

In Abschn. 7.4 wurde gezeigt, wie man ein Bayes-Netz manuell aufbauen kann. Nun werden wir Algorithmen zum induktiven Lernen von Bayes-Netzen vorstellen. Ähnlich wie bei den bisher beschriebenen Lernverfahren wird aus einer Datei mit Trainingsdaten ein Bayes-Netz automatisch erzeugt. Dieser Prozess wird typischerweise zerlegt in zwei Teile:

1. Lernen der Netzwerkstruktur Bei vorgegebenen Variablen wird aus den Trainingsdaten die Netzwerktopologie generiert. Dieser erste Schritt ist der mit Abstand schwierigere und verdient im Folgenden eine genauere Betrachtung.

2. Lernen der bedingten Wahrscheinlichkeiten Bei bekannter Netzwerktopologie müssen die CPTs mit Werten gefüllt werden. Wenn genügend Trainingsdaten verfügbar sind, können durch Zählen von Häufigkeiten in den Daten alle benötigten bedingten Wahrscheinlichkeiten geschätzt werden. Dieser Schritt ist relativ leicht automatisierbar.

Anhand eines einfachen Verfahrens aus [Jen01] wollen wir nun die Idee des Lernens von Bayes-Netzen vermitteln.

8.6.1 Lernen der Netzwerkstruktur

Bei der Entwicklung eines Bayes-Netzes (siehe Abschn. 7.4.6) muss unter anderem die kausale Abhängigkeit der Variablen berücksichtigt werden, um ein einfaches Netz mit guter Qualität zu erhalten. Der menschliche Entwickler greift hierfür auf Hintergrundwissen

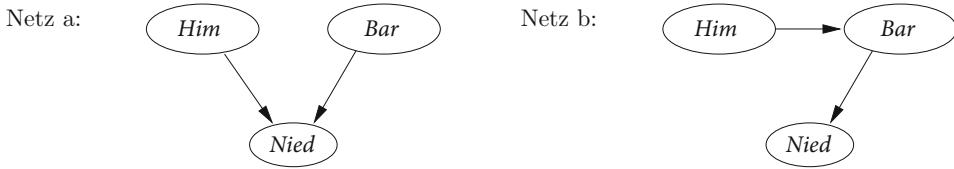


Abb. 8.28 Zwei Bayes-Netze zur Modellierung des Wettervorhersage-Beispiels aus Aufgabe 7.3

zurück, über das die Maschine nicht verfügt. Daher kann dieses Vorgehen nicht einfach automatisiert werden.

Das Finden einer optimalen Struktur für ein Bayes-Netz lässt sich als klassisches Suchproblem formulieren. Gegeben ist eine Menge von Variablen V_1, \dots, V_n und eine Datei mit Trainingsdaten. Gesucht ist eine zyklusfreie Menge von gerichteten Kanten auf den Knoten V_1, \dots, V_n , also ein gerichteter azyklischer Graph (engl. directed acyclic graph, DAG), welcher die den Daten zugrunde liegende Wahrscheinlichkeitsverteilung möglichst gut wiedergibt.

Betrachten wir zuerst den Suchraum. Die Zahl unterschiedlicher DAGs wächst überexponentiell mit der Zahl der Knoten. Bei 5 Knoten gibt es 29281 und bei 9 Knoten schon etwa 10^{15} verschiedene DAGs [MDBM00]. Damit ist eine uninformede kombinatorische Suche (siehe Abschn. 6.2) im Raum aller Graphen mit einer gegebenen Menge von Variablen aussichtslos, wenn die Zahl der Variablen wächst. Es müssen daher heuristische Verfahren verwendet werden. Damit stellt sich die Frage nach einer Bewertungsfunktion für Bayes-Netze. Denkbar wäre eine Messung des Klassifikationsfehlers des Netzes bei Anwendung auf eine Menge von Testdaten wie dies zum Beispiel bei C4.5 (siehe Abschn. 8.4) erfolgt. Hierzu müssten aber die Wahrscheinlichkeiten, die das Bayes-Netz berechnet, zuerst auf eine Entscheidung abgebildet werden.

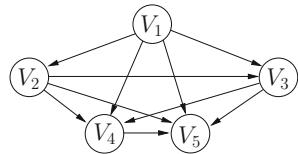
Eine direkte Messung der Qualität des Netzes kann erfolgen über die Wahrscheinlichkeitsverteilung. Wir nehmen an, dass wir vor dem Aufbau des Netzes aus den Daten die Verteilung bestimmen (schätzen) können. Dann starten wir die Suche im Raum aller DAGs, schätzen für jeden DAG (das heißt jedes Bayes-Netz) anhand der Daten die Werte der CPTs, berechnen daraus die Verteilung und vergleichen diese mit der aus den Daten bekannten Verteilung. Für den Vergleich von Verteilungen wird offenbar ein Abstandsmaß benötigt.

Betrachten wir das Beispiel zur Wettervorhersage aus Aufgabe 7.3 mit den drei Variablen *Him*, *Bar*, *Nied*, und der Verteilung

$$P(Him, Bar, Nied) = (0,40, 0,07, 0,08, 0,10, 0,09, 0,11, 0,03, 0,12).$$

In Abb. 8.28 sind zwei Bayes-Netze dargestellt, die wir nun bezüglich ihrer Qualität vergleichen wollen. Jedes dieser Netze macht eine Unabhängigkeitsannahme, die es zu prüfen gilt, indem wir jeweils die Verteilung des Netzes bestimmen und dann mit der Originalverteilung vergleichen (siehe Aufgabe 8.16).

Abb. 8.29 Das maximale Netz mit fünf Variablen und Kanten (V_i, V_j) , welche die Bedingung $i < j$ erfüllen



Da die Verteilungen bei fest vorgegebenen Variablen offenbar einen Vektor fester Länge darstellen, können wir zur Definition des Abstandes von Verteilungen die euklidische Norm der Differenz der beiden Vektoren berechnen. Wir definieren

$$d_q(\mathbf{x}, \mathbf{y}) = \sum_i (x_i - y_i)^2$$

als Summe der Abstandsquadrate der Vektorkomponenten und berechnen den Abstand $d_q(\mathbf{P}_a, \mathbf{P}) = 0,0029$ der Verteilung \mathbf{P}_a des Netzes a von der Originalverteilung. Für Netz b berechnen wir $d_q(\mathbf{P}_b, \mathbf{P}) = 0,014$. Offenbar ist Netz a die bessere Approximation der Verteilung. Oft wird statt dem quadratischen Abstand auch der so genannte Kullback-Leibler-Abstand

$$d_k(\mathbf{x}, \mathbf{y}) = \sum_i y_i (\log_2 y_i - \log_2 x_i),$$

ein informationstheoretisches Maß, verwendet. Damit berechnen wir $d_k(\mathbf{P}_a, \mathbf{P}) = 0,017$ und $d_k(\mathbf{P}_b, \mathbf{P}) = 0,09$ und kommen zu dem gleichen Ergebnis wie zuvor. Es ist zu erwarten, dass Netze mit vielen Kanten die Verteilung besser approximieren als solche mit wenigen Kanten. Werden aber alle Kanten in das Netz eingebaut, so wird es sehr unübersichtlich, und es besteht die Gefahr der Überanpassung wie auch bei vielen anderen Lernverfahren. Zur Vermeidung der Überanpassung gibt man kleinen Netzen ein größeres Gewicht mittels einer heuristischen Bewertungsfunktion

$$f(N) = \text{Größe}(N) + w \cdot d_k(\mathbf{P}_N, \mathbf{P}).$$

Hierbei ist $\text{Größe}(N)$ die Anzahl der Einträge in den CPTs und \mathbf{P}_N die Verteilung des Netzes N. w ist ein Gewichtungsfaktor, der manuell angepasst werden muss.

Der Lernalgorithmus für Bayes-Netze berechnet also für viele verschiedene Netze deren heuristische Bewertung $f(N)$ und wählt dann das Netz mit dem kleinsten Wert. Wie schon erwähnt, besteht die Schwierigkeit in der Reduktion des Suchraumes der zu prüfenden Netztopologien. Als einfaches Verfahren bietet es sich an, ausgehend von einer (zum Beispiel kausalen) Ordnung der Variablen V_1, \dots, V_n nur diejenigen gerichteten Kanten (V_i, V_j) in das Netz einzuziehen, für die $i < j$. Gestartet wird mit dem maximalen Modell, das diese Bedingung erfüllt. Für fünf geordnete Variablen ist dieses Netz in Abb. 8.29 dargestellt.

Nun wird, zum Beispiel im Sinne einer gierigen Suche (vgl. Abschn. 6.3.1), eine Kante nach der anderen entfernt, so lange bis der Wert f nicht mehr weiter abnimmt.

Dieses Verfahren ist in dieser Form für größere Netze nicht praktisch einsetzbar. Gründe hierfür sind der große Suchraum, das manuelle Tuning des Gewichtes w , aber auch der hier notwendige Vergleich mit einer Zielverteilung P , denn diese könnte schlicht zu groß werden, beziehungsweise die verfügbare Datenmenge ist zu klein.

Tatsächlich ist die Forschung zum Lernen von Bayes-Netzen noch in vollem Gange, und es gibt eine große Zahl von vorgeschlagenen Algorithmen, zum Beispiel den EM-Algorithmus (siehe Abschn. 8.9.2), die Markov-Chain-Monte-Carlo-Methoden oder das Gibbs-Sampling [DHS01, Jor99, Jen01, HTF09]. Neben dem hier vorgestellten Batch-Learning, bei dem einmal aus der gesamten Datenmenge das Netz generiert wird, gibt es auch inkrementelle Verfahren, bei denen jeder einzelne neue Fall zur Verbesserung des Netzes verwendet wird. Auch gibt es Implementierungen, zum Beispiel in Hugin (<http://www.hugin.com>) oder Bayesware (<http://www.bayesware.com>).

8.7 Der Naive-Bayes-Klassifizierer

In Abb. 7.14 wurde die Appendizitisdiagnose als Bayes-Netz modelliert. Da vom Diagnoseknoten *Bef4* nur Pfeile ausgehen und keine dort enden, muss zur Beantwortung einer Diagnoseanfrage die Bayes-Formel verwendet werden. Man berechnet für die Symptome S_1, \dots, S_n und den k -wertigen Befund B mit den Werten b_1, \dots, b_k die Wahrscheinlichkeit

$$P(B | S_1, \dots, S_n) = \frac{P(S_1, \dots, S_n | B) \cdot P(B)}{P(S_1, \dots, S_n)},$$

für den Befund gegeben die Symptome des Patienten. Im schlimmsten Fall, das heißt, wenn es keine Unabhängigkeiten gäbe, müssten für alle Kombinationen von allen Symptomvariablen und B alle $20.643.840$ Wahrscheinlichkeiten der Verteilung $P(S_1, \dots, S_n, B)$ bestimmt werden. Dafür würde man eine riesige Datenbank benötigen. Im Fall des Bayes-Netzes für LEXMED reduzierte sich die Zahl der benötigten Werte (in den CPTs) auf 521. Das Netz lässt sich aber noch weiter vereinfachen, indem man annimmt, alle Symptomvariablen sind bedingt unabhängig gegeben B , das heißt

$$P(S_1, \dots, S_n | B) = P(S_1 | B) \cdot \dots \cdot P(S_n | B).$$

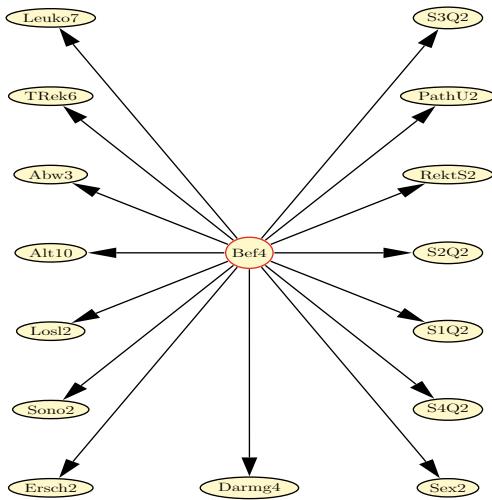
Das Bayes-Netz für Appendizitis vereinfacht sich dann auf den in Abb. 8.30 dargestellten Stern.

Damit erhält man die Formel

$$P(B | S_1, \dots, S_n) = \frac{P(B) \prod_{i=1}^n P(S_i | B)}{P(S_1, \dots, S_n)}. \quad (8.8)$$

Die berechneten Wahrscheinlichkeiten werden in eine Entscheidung überführt durch den einfachen Naive-Bayes-Klassifizierer, der aus allen Werten b_i von B den mit maximalem

Abb. 8.30 Bayes-Netz für die LEXMED-Anwendung unter der Annahme, alle Symptome sind bedingt unabhängig gegeben der Befund



$P(B = b_i | S_1, \dots, S_n)$ auswählt. Das heißt, er bestimmt

$$b_{\text{Naive-Bayes}} = \underset{i \in \{1, \dots, k\}}{\operatorname{argmax}} P(B = b_i | S_1, \dots, S_n).$$

Da der Nenner in (8.8) konstant ist, kann er bei der Maximierung weggelassen werden, was zur **Naive-Bayes-Formel**

$$b_{\text{Naive-Bayes}} = \underset{i \in \{1, \dots, k\}}{\operatorname{argmax}} P(B = b_i) \prod_{j=1}^n P(S_j | B)$$

führt. Da einige Knoten nun weniger Vorgänger haben, verringert sich die Anzahl der zur Beschreibung der LEXMED-Verteilung benötigten Werte in den CPTs nun nach (7.22) auf

$$6 \cdot 4 + 5 \cdot 4 + 2 \cdot 4 + 9 \cdot 4 + 3 \cdot 4 + 10 \cdot (1 \cdot 4) + 1 = 141.$$

Für ein medizinisches Diagnosesystem wie LEXMED wäre diese Vereinfachung nicht mehr akzeptabel. Aber für Aufgaben mit vielen unabhängigen Variablen ist Naive-Bayes teilweise sogar sehr gut geeignet, wie wir am Beispiel der Textklassifikation sehen werden.

Die Naive-Bayes-Klassifikation ist übrigens gleich ausdrucksstark wie die in Abschn. 7.3.1 erwähnten linearen Score-Systeme (siehe Aufgabe 8.17). Das heißt, allen Scores liegt die Annahme zugrunde, dass alle Symptome bedingt unabhängig gegeben der Befund sind. Trotzdem wird in der Medizin heute immer noch mit Scores gearbeitet. Obwohl aus einer besseren, repräsentativen Datenbasis erzeugt, besitzt der in Abb. 7.10 mit LEXMED

vergleichene Ohmann-Score schlechtere Diagnosequalitäten. Grund hierfür ist sicherlich die beschränkte Ausdrucksfähigkeit der Scores. Zum Beispiel ist es, wie schon erwähnt, bei Scores, wie auch bei Naive-Bayes, nicht möglich, Abhängigkeiten zwischen den Symptomen zu modellieren.

Schätzen von Wahrscheinlichkeiten

Betrachtet man die Naive-Bayes-Formel in (8.8), so erkennt man, dass der ganze Ausdruck null wird, sobald einer der Faktoren $P(S_i | B)$ auf der rechten Seite null wird. Theoretisch ist hier alles in Ordnung. In der Praxis jedoch kann dies zu sehr unangenehmen Effekten führen, wenn die $P(S_i | B)$ klein sind, denn diese werden geschätzt durch Abzählen von Häufigkeiten und Einsetzen in

$$P(S_i = x | B = y) = \frac{|S_i = x \wedge B = y|}{|B = y|}.$$

Angenommen, für die Variable S_i ist $P(S_i = x | B = y) = 0,01$ und es gibt 40 Trainingsfälle mit $B = y$. Dann gibt es mit hoher Wahrscheinlichkeit keinen Trainingsfall mit $S_i = x$ und $B = y$ und es wird $P(S_i = x | B = y) = 0$ geschätzt. Für einen anderen Wert $B = z$ mögen die Verhältnisse ähnlich gelagert sein, aber die Schätzung ergibt für alle $P(S_i = x | B = z)$ Werte größer als null. Damit wird der Wert $B = z$ immer bevorzugt, was nicht der tatsächlichen Wahrscheinlichkeitsverteilung entspricht. Daher wird beim Schätzen von Wahrscheinlichkeiten die Formel

$$P(A | B) \approx \frac{|A \wedge B|}{|B|} = \frac{n_{AB}}{n_B}$$

ersetzt durch

$$P(A | B) \approx \frac{n_{AB} + mp}{n_B + m},$$

wobei $p = P(A)$, also die A-priori-Wahrscheinlichkeit für A ist und m eine frei wählbare Konstante, die auch als „äquivalente Datenmächtigkeit“ bezeichnet wird. Je größer m wird, desto größer das Gewicht der A-priori-Wahrscheinlichkeit im Vergleich zu dem aus der gemessenen Häufigkeit bestimmten Wert.

8.7.1 Textklassifikation mit Naive-Bayes

Sehr erfolgreich und weit verbreitet ist Naive-Bayes heute bei der Textklassifikation. Die primäre und gleichzeitig auch sehr wichtige Anwendung ist das automatische Filtern von Email in erwünschte und nicht erwünschte, so genannte Spam-Mails. In Spam-Filtern wie zum Beispiel SpamAssassin [Sch04] wird neben anderen Methoden ein Naive-Bayes-Klassifizierer verwendet, der lernt, erwünschte Mails von Spam zu trennen. SpamAssassin

ist ein hybrides System, welches eine erste Filterung anhand von Black- und Whitelists durchführt. Blacklists sind Listen gesperrter Emailadressen von Spamversendern, deren Mails immer gelöscht werden und Whitelists sind solche mit Absendern, deren Mails immer ausgeliefert werden. Nach dieser Vorfilterung wird von den verbleibenden Emails der eigentliche Inhalt, das heißt der Text, der Email durch den Naive-Bayes-Klassifizierer klassifiziert. Der ermittelte Klassenwert wird dann zusammen mit anderen Attributen aus dem Kopf der Email wie etwa die Absenderdomain, der MIME-Typ, etc. durch einen Score bewertet und schließlich gefiltert.

Ganz wichtig hierbei ist die Lernfähigkeit des Naive-Bayes-Filters. Dazu muss der Benutzer am Anfang eine größere Anzahl Emails manuell als erwünscht oder Spam klassifizieren. Dann wird der Filter trainiert. Um „auf dem Laufenden“ zu bleiben, muss der Filter regelmäßig neu trainiert werden. Dazu sollte der Benutzer alle vom Filter falsch klassifizierten Emails korrekt klassifizieren, das heißt in entsprechenden Ordnern ablegen. Der Filter wird mit diesen Mails dann immer wieder nachtrainiert.

Neben der Spam-Filterung gibt es noch viele weitere Anwendungen für die automatische Textklassifikation. Ganz wichtig ist heute das Ausfiltern von unerwünschten Beiträgen in Internet-Diskussionsforen sowie das Aufspüren von Webseiten mit kriminellen Inhalten wie etwa rechtsradikale oder terroristische Aktivitäten, Kinderpornographie oder Rassistische Inhalte. Auch können hiermit Suchmaschinen den Wünschen des Benutzers angepasst werden, um die gefundenen Hits besser zu klassifizieren. Im industriellen und wissenschaftlichen Umfeld steht die unternehmensweite Suche in Datenbanken oder in der Literatur nach relevanten Informationen im Vordergrund. Durch die Lernfähigkeit passt sich der Filter an die Gewohnheiten und Wünsche des jeweiligen Benutzers individuell an.

Die Anwendung von Naive-Bayes auf die Textanalyse wollen wir an einem kurzen Beispieltext von Alan Turing aus [Tur50] vorstellen:

Wir dürfen hoffen, dass Maschinen vielleicht einmal auf allen rein intellektuellen Gebieten mit dem Menschen konkurrieren. Aber mit welchem sollte man am besten beginnen? Viele glauben, dass eine sehr abstrakte Tätigkeit, beispielsweise das Schachspielen, am besten geeignet wäre. Ebenso kann man behaupten, dass es das beste wäre, Maschinen mit den besten Sinnesorganen auszustatten, die überhaupt für Geld zu haben sind, und sie dann zu lehren, englisch zu verstehen und zu sprechen. Dieser Prozess könnte sich wie das normale Unterrichten eines Kindes vollziehen. Dinge würden erklärt und benannt werden, usw. Wiederum weiß ich nicht, welches die richtige Antwort ist, aber ich meine, dass man beide Ansätze erproben sollte.

Es seien Texte wie der angegebene in die zwei Klassen „*I*“ für interessant und „*-I*“ für uninteressant einzuteilen. Dazu existiere eine Datenbank schon klassifizierter Texte. Welche Attribute sollen nun verwendet werden? In einem klassischen Ansatz zum Bau eines Bayes-Netzes würde man eine Menge von Attributen wie etwa die Länge des Textes, die mittlere Länge der Sätze, die relative Häufigkeit bestimmter Satzzeichen, die Häufigkeit einiger wichtiger Wörter wie zum Beispiel „ich“, „Maschinen“, etc. definieren. Bei der

Klassifikation mittels Naive-Bayes hingegen wird ein überraschend primitives Verfahren 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 Worte erlaubt. Nun müssen für die Klassen I und $\neg I$ die Werte

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \prod_{i=1}^n P(s_i | I) \quad (8.9)$$

sowie $P(\neg I | s_1, \dots, s_n)$ berechnet und dann die Klasse mit dem maximalen Wert gewählt werden. In obigem Beispiel mit insgesamt 107 Worten ergibt dies

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \cdot P(s_1 = „\text{Wir}“ | I) \cdot P(s_2 = „\text{dürfen}“ | I) \cdot \dots \cdot P(s_{107} = „\text{sollte}“ | I)$$

und

$$\begin{aligned} P(\neg I | s_1, \dots, s_n) &= c \cdot P(\neg I) \cdot P(s_1 = „\text{Wir}“ | \neg I) \\ &\quad \cdot P(s_2 = „\text{dürfen}“ | \neg I) \cdot \dots \cdot P(s_{107} = „\text{sollte}“ | \neg I). \end{aligned}$$

Das Lernen ist hier ganz einfach. Es müssen lediglich auf den beiden Klassen von Texten die bedingten Wahrscheinlichkeiten $P(s_i | I)$ und $P(s_i | \neg I)$ sowie die A-priori-Wahrscheinlichkeiten $P(I), P(\neg I)$ berechnet werden. Nun nehmen wir noch an, dass die $P(s_i | I)$ nicht von der Position im Text abhängen. Das heißt zum Beispiel, dass

$$P(s_{61} = „\text{und}“ | I) = P(s_{69} = „\text{und}“ | I) = P(s_{86} = „\text{und}“ | I).$$

Wir können also als Wahrscheinlichkeit für das Vorkommen von „und“ an einer beliebigen Stelle den Ausdruck $P(\text{und} | I)$ mit der neuen binären Variablen *und* verwenden.

Damit kann die Implementierung etwas beschleunigt werden, wenn wir für jedes im Text vorkommende Wort w_i seine Häufigkeit n_i ermitteln und die zu (8.9) äquivalente Formel

$$P(I | s_1, \dots, s_n) = c \cdot P(I) \prod_{i=1}^l P(w_i | I)^{n_i} \quad (8.10)$$

verwenden. Man beachte, dass der Index i im Produkt nun nur noch bis zur Zahl l unterschiedlicher im Text vorkommender Worte läuft.

Trotz seiner Einfachheit liefert Naive-Bayes bei der Textklassifikation hervorragende Ergebnisse. Mit Naive-Bayes arbeitende Spam-Filter erreichen Fehlerraten von weit unter einem Prozent. Die Systeme DSPAM und CRM114 können sogar so gut trainiert werden, dass sie nur noch eine von 7000 beziehungsweise 8000 Emails falsch klassifizieren [Zdz05]. Dies entspricht einer Korrektheit von fast 99,99 %.

8.8 One-Class-Learning

Für das Lernen mit Lehrer müssen bei Klassifikationsaufgaben alle Trainingsdaten mit einem Klassen-Label versehen werden. Es gibt aber Anwendungen, bei denen nur Daten einer Klasse verfügbar sind. Ein klassisches Beispiel hierfür ist die Diagnose technischer Anlagen. Es soll hier erkannt werden ob die Anlage einen Defekt hat. Dies ist ein klassisches Zweiklassenproblem. In den Trainingsdaten muss für jeden Datenpunkt der Zustand der Anlage (gut oder fehlerhaft) manuell angegeben werden muss. Das Training des Klassifizierers erfolgt in der Praxis zum Beispiel bei der Inbetriebnahme der Anlage oder auch später bei Bedarf während des laufenden Betriebs. Das Erfassen von Daten im fehlerfreien Zustand ist unproblematisch. Dagegen erweist sich die Sammlung von Daten einer Anlage mit Fehlern als problematisch aus folgenden Gründen:

- Die Messung an einer bewusst mit einem Defekt versehenen Anlage in der Produktion ist mit hohen Kosten verbunden, denn die Messung führt zu teuren Ausfallzeiten.
- Die Messung an einer defekten Anlage mit real auftretenden Fehlern ist oft gar nicht möglich, denn bei der Inbetriebnahme der Anlage sind die später auftretenden Fehlerarten eventuell noch unbekannt.
- Kein Ingenieur weiß im Voraus bei einer neuen Anlage, welche Arten von Fehlern im Betrieb auftreten werden. Werden nun in der Trainingsphase Messungen mit einigen wenigen Fehlern durchgeführt, so kann dies zu schlechten Klassifikationsergebnissen führen. Wenn die zum Training verwendeten Fehler nicht repräsentativ für die Anlage sind, wird die so gelernte, die Klassen trennende Hyperebene im Merkmalsraum oft zu falsch positiven Klassifikationen führen.

In solchen Fällen ist das klassische Trainieren eines Klassifizierers für zwei Klassen nicht möglich. Stattdessen wird das One-Class-Learning verwendet, das beim Training mit Daten einer Klasse auskommt.

Es stehen also keine negativen Daten zur Verfügung. Positive Daten, das heißt vom fehlerfreien Betrieb sind jedoch in ausreichender Menge verfügbar. Es wird also ein Lernverfahren benötigt, das basierend auf den fehlerfreien Daten möglichst alle fehlerfreien Betriebszustände der Anlage erfasst und alle anderen als negativ klassifiziert.

Für diesen Zweck gibt es unter dem Namen One-Class-Learning eine Anzahl verschiedener Algorithmen [Tax01], unter anderem die Nearest Neighbour Data Description (NNDD) und die Support Vector Data Description (SVDD, siehe auch Abschn. 9.6). Diese und verwandte Verfahren sind in der Statistik unter dem Namen Outlier Detection bekannt.

8.8.1 Nearest Neighbour Data Description

Die Nearest Neighbour Data Description gehört wie die Nearest Neighbour-Methode zur Klasse der Lazy-Learning Verfahren. Beim Lernen werden hier die Merkmals-Vektoren nur

normalisiert und gespeichert. Daher das Attribut „Lazy“ beim Lernen. Die Normalisierung jedes einzelnen Merkmals ist notwendig, um unterschiedlichen Merkmalen gleiches Gewicht zu geben. Ohne Normalisierung würde zum Beispiel ein Merkmal mit Wertebereich $[0, 10^{-4}]$ im Vergleich zu einem Merkmal mit Wertebereich $[-10.000, +10.000]$ im Rauschen untergehen. Daher werden alle Merkmale linear auf das Intervall $[0, 1]$ skaliert.⁹ Der eigentliche Nearest-Neighbour-Algorithmus kommt erst bei der Klassifikation zum Einsatz.

Gegeben sei eine Trainingsdatenmenge $X = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ bestehend aus n Merkmalvektoren. Ein neuer, zu klassifizierende Punkt \mathbf{q} wird akzeptiert, wenn

$$D(\mathbf{q}, \text{NN}(\mathbf{q})) \leq \gamma \bar{D}, \quad (8.11)$$

das heißtt, wenn der Abstand zu einem nächsten Nachbarn nicht größer ist als $\gamma \bar{D}$. Hierbei ist $D(\mathbf{x}, \mathbf{y})$ eine Metrik, zum Beispiel wie hier verwendet die Euklidische Norm. Die Funktion

$$\text{NN}(\mathbf{q}) = \operatorname{argmin}_{\mathbf{x} \in X} \{D(\mathbf{q}, \mathbf{x})\}$$

ermittelt einen nächsten Nachbarn von \mathbf{q} und

$$\bar{D} = \frac{1}{n} \sum_{i=1}^n D(\mathbf{x}_i, \text{NN}(\mathbf{x}_i))$$

ist der mittlere Abstand der nächsten Nachbarn zu allen Datenpunkten in X . Zur Berechnung von \bar{D} wird für jeden Datenpunkt der Abstand zu seinem nächsten Nachbarn berechnet. \bar{D} ist dann das arithmetische Mittel aller so berechneten Abstände. Man könnte in (8.11) mit $\gamma = 1$ arbeiten, würde allerdings suboptimale Klassifikationsergebnisse bekommen. Besser ist es, den Parameter γ mittels Kreuzvalidierung (Abschn. 8.5) so zu bestimmen, dass die Fehlerrate des NNDD-Klassifizierers möglichst klein wird.

Das hier verwendete NNDD-Verfahren ist eine Modifikation des in [Tax01] verwendeten Verfahrens NNDD_T. Dort wird ein Punkt \mathbf{q} akzeptiert, wenn der Abstand zum nächsten Nachbarn in den Trainingsdaten nicht größer ist als der Abstand des gefundenen nächsten Nachbarn zu dessen nächstem Nachbarn in den Trainingsdaten, das heißtt wenn

$$D(\mathbf{q}, \text{NN}(\mathbf{q})) \leq D(\text{NN}(\mathbf{q}), \text{NN}(\text{NN}(\mathbf{q}))). \quad (8.12)$$

Diese Formel hat mehrere Nachteile im Vergleich zu Ungleichung (8.11). Erstens ergeben sich bei Verwendung von NNDD_T (Ungleichung (8.12)) intuitiv unschöne Klassentrennlinien, wie man in Abb. 8.31 rechts an einem einfachen zweidimensionalen Beispiel gut

⁹ Eine Skalierung der Merkmale ist für viele Verfahren des maschinellen Lernens notwendig oder vorteilhaft.

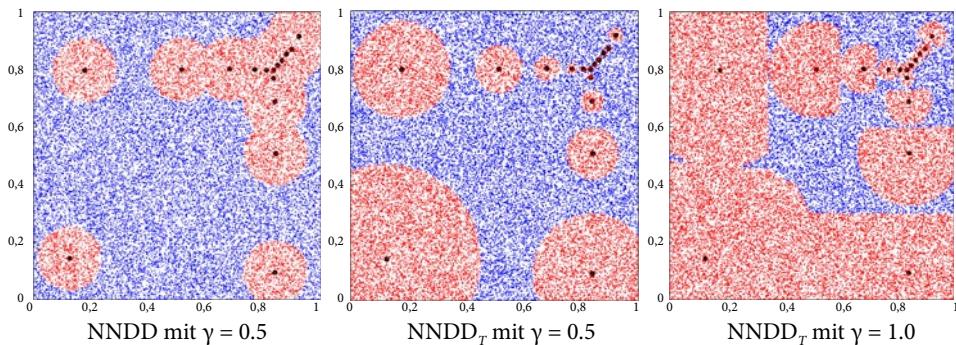


Abb. 8.31 Auf eine Menge von 16 ausgewählten zweidimensionalen Datenpunkten (schwarze Punkte) wurde NNDD und NNDD_T angewendet. Für jeweils 10.000 zufällig gewählte Punkte wurde deren Klassenzugehörigkeit bestimmt. Die rot markierten Punkte wurden als positiv klassifiziert, d. h. der Klasse der Trainingsdaten zugeordnet, die blau markierten dagegen als negativ

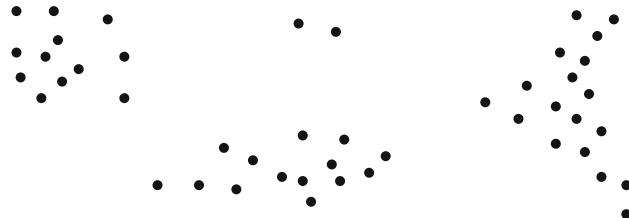
erkennen kann. In der Mitte von Abb. 8.31 sieht man, dass der Datenpunkt links unten einen großen Einzugsbereich definiert, obwohl er weit von allen anderen Datenpunkten entfernt ist. NNDD mit (8.11) hingegen liefert die in Abb. 8.31 links gezeigten kreisförmigen Trennlinien mit konstantem Radius für die Klassentrennung. Diese intuitive Vermutung wird auch durch praktische Experimente bestätigt.

8.9 Clustering

Wenn man in einer Suchmaschine nach dem Begriff „Decke“ sucht, so werden Treffer wie etwa „Microfaser-Decke“, „unter der Decke“, etc. angezeigt, genauso wie „Stuck an der Decke“, „Wie streiche ich eine Decke“, etc. Bei der Menge der gefundenen Textdokumente handelt es sich um zwei deutlich von einander unterscheidbare **Cluster**. Noch ist es so, dass zum Beispiel Google die Treffer unstrukturiert auflistet. Schöner wäre es aber, wenn die Suchmaschine die Cluster trennen und dem Benutzer entsprechend präsentieren würde, denn dieser interessiert sich normalerweise nur für die Treffer in einem der gefundenen Cluster.

Die Besonderheit beim **Clustering** im Unterschied zum Lernen mit Lehrer ist die, dass die Trainingsdaten nicht klassifiziert sind. Die Vorstrukturierung der Daten durch den Lehrer fehlt hier also. Genau um das Finden von Strukturen geht es hier. Es sollen im Raum der Trainingsdaten Häufungen von Daten gefunden werden, wie sie zum Beispiel in Abb. 8.32 deutlich erkennbar sind. In einem Cluster ist der Abstand benachbarter Punkte typischerweise kleiner als der Abstand zwischen Punkten aus verschiedenen Clustern. Ganz wichtig und grundlegend ist daher die Wahl eines geeigneten Abstandsmaßes für Punkte, das heißt für zu gruppierende Objekte und für Cluster. Wie bisher nehmen wir im Folgenden an, jedes Datenobjekt wird durch einen Vektor numerischer Attribute beschrieben.

Abb. 8.32 Einfaches zweidimensionales Beispiel mit vier deutlich getrennten Clustern



8.9.1 Abstandsmaße

Je nach der Anwendung werden die verschiedensten Abstandsmaße für die Distanz d zwischen zwei Vektoren \mathbf{x} und \mathbf{y} im \mathbb{R}^n definiert. Am gebräuchlichsten ist der Euklidische Abstand

$$d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

Etwas einfacher ist die Summe der Abstandsquadrate

$$d_q(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n (x_i - y_i)^2,$$

die für Algorithmen, bei denen nur Abstände verglichen werden, äquivalent zum Euklidischen Abstand ist (Aufgabe 8.20). Aber auch der schon bekannte Manhattan-Abstand

$$d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$$

sowie der auf der Maximumnorm basierende Abstand der maximalen Komponente

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1, \dots, n} |x_i - y_i|$$

werden verwendet. Bei der Textklassifikation wird häufig die normierte Projektion der beiden Vektoren aufeinander, das heißt das normierte Skalarprodukt

$$\frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| |\mathbf{y}|}$$

berechnet, wobei $|\mathbf{x}|$ die Euklidische Norm von \mathbf{x} ist. Da diese Formel ein Maß für die Ähnlichkeit der beiden Vektoren darstellt, wird als Abstandsmaß zum Beispiel der Kehrwert

$$d_s(\mathbf{x}, \mathbf{y}) = \frac{|\mathbf{x}| |\mathbf{y}|}{\mathbf{x} \cdot \mathbf{y}}$$

verwendet oder bei allen Vergleichen „>“ und „<“ vertauscht. Bei der Suche nach einem Text werden die Merkmale x_1, \dots, x_n als Komponenten des Vektors \mathbf{x} , ähnlich wie bei Naive-Bayes, wie folgt berechnet. Bei einem Wörterbuch mit zum Beispiel 50.000 Wörtern hat der Vektor die Länge 50.000 und der Wert x_i ist gleich der Häufigkeit des i -ten Wörterbuch-Wortes im Text. Da in solch einem Vektor normalerweise fast alle Komponenten null sind, sind auch bei der Berechnung des Skalarproduktes fast alle Summanden gleich null. Durch Ausnutzung derartiger Informationen kann die Implementierung stark beschleunigt werden (Aufgabe 8.21).

8.9.2 k -Means und der EM-Algorithmus

Immer dann, wenn die Zahl der Cluster schon im Voraus bekannt ist, kann das **k -Means**-Verfahren zum Einsatz kommen, bei dem, wie der Name schon sagt, k Cluster durch ihre Mittelwerte definiert werden. Zuerst werden die k Clustermittelpunkte μ_1, \dots, μ_k zufällig oder manuell initialisiert. Dann werden wiederholt die folgenden beiden Schritte ausgeführt:

- Klassifikation aller Daten zum nächsten Clustermittelpunkt
- Neuberechnung der Clustermittelpunkte

Als Algorithmus ergibt sich das Schema

```

K-MEANS( $\mathbf{x}_1, \dots, \mathbf{x}_n, k$ )
initialisiere  $\mu_1, \dots, \mu_k$  (z.B. zufällig)
Repeat
    Klassifiziere  $\mathbf{x}_1, \dots, \mathbf{x}_n$  zum jeweils nächsten  $\mu_i$ 
    Berechne  $\mu_1, \dots, \mu_k$  neu
Until keine Änderung in  $\mu_1, \dots, \mu_k$ 
Return( $\mu_1, \dots, \mu_k$ )

```

Die Berechnung des Clustermittelpunktes μ für Punkte $\mathbf{x}_1, \dots, \mathbf{x}_l$ erfolgt durch

$$\mu = \frac{1}{l} \sum_{i=1}^l \mathbf{x}_i.$$

Für den Fall von zwei Klassen ist der Ablauf in Abb. 8.33 an einem Beispiel dargestellt. Man erkennt, wie sich nach drei Iterationen die anfangs zufällig gewählten Klassenzentren stabilisieren. Obwohl es für dieses Verfahren keine Konvergenzgarantie gibt, konvergiert es meist recht schnell. Das heißt, die Zahl der Iterationsschritte ist typischerweise viel kleiner als die Zahl der Datenpunkte. Die Komplexität ist $O(ndkt)$, wobei n die Gesamtzahl der Punkte, d die Dimension des Merkmalsraumes und t die Zahl der Iterationsschritte ist.

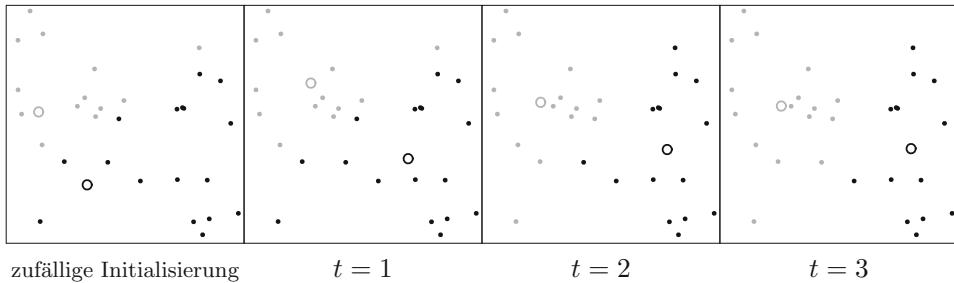


Abb. 8.33 k -Means mit zwei Klassen ($k = 2$) angewendet auf 30 Datenpunkte. Ganz links die Datenmenge mit den initialen Zentren und nach rechts die Cluster nach jeder Iteration. Nach drei Iterationen ist die Konvergenz erreicht

In vielen Fällen stellt die Notwendigkeit, die Zahl der Klassen vorzugeben, eine unangenehme Einschränkung dar. Daher werden wir als nächstes ein Verfahren vorstellen, das flexibler ist. Zuvor jedoch sei noch auf den **EM-Algorithmus** verwiesen, der eine stetige Variante von k -Means darstellt, denn er macht keine harte Zuordnung der Daten zu den Klassen, sondern liefert für jeden Punkt die Wahrscheinlichkeiten für die Zugehörigkeit zu den verschiedenen Klassen. Man geht hierbei davon aus, dass die Art der Wahrscheinlichkeitsverteilung der Daten bekannt ist. Oft wird die Normalverteilung verwendet. Aufgabe des EM-Algorithmus ist es nun, die Parameter (Mittelwert μ_i und Kovarianzmatrix Σ_i der k mehrdimensionalen Normalverteilungen) für jeden Cluster zu bestimmen. Ähnlich wie bei k -Means werden wiederholt die beiden folgenden Schritte ausgeführt:

Expectation: Für jeden Datenpunkt wird berechnet, mit welcher Wahrscheinlichkeit $P(C_j | \mathbf{x}_i)$ er zu jedem der Cluster gehört.

Maximization: Unter Verwendung der neu berechneten Wahrscheinlichkeiten werden die Parameter der Verteilung neu berechnet.

Dadurch wird ein weicheres Clustering erreicht, das in vielen Fällen zu besseren Ergebnissen führt. Dieses Abwechseln zwischen Expectation und Maximization gibt dem Algorithmus seinen Namen. Der EM-Algorithmus wird neben dem Clustering zum Beispiel für das Lernen von Bayes-Netzen eingesetzt [DHS01].

8.9.3 Hierarchisches Clustering

Beim hierarchischen Clustering beginnt man mit n Clustern bestehend aus jeweils einem Punkt. Dann werden so lange jeweils die beiden nächsten Nachbarcluster vereinigt, bis alle Punkte in einem einzigen Cluster vereinigt sind oder ein Abbruchkriterium erreicht ist. Wir erhalten das Schema

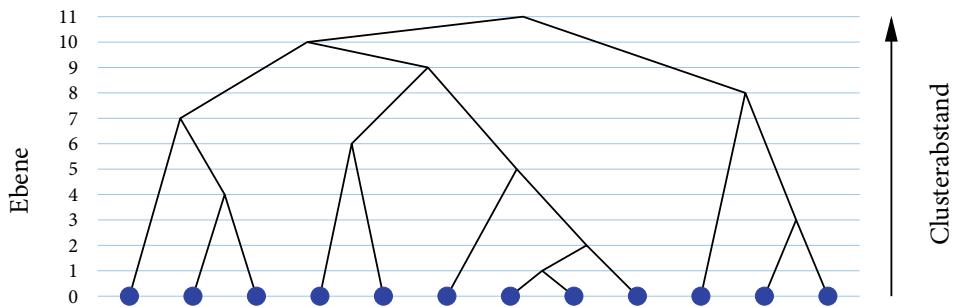


Abb. 8.34 Beim hierarchischen Clustering werden in jedem Schritt die zwei Cluster mit dem kleinsten Abstand vereinigt

HIERARCHISCHES-CLUSTERING (x_1, \dots, x_n)

initialisiere $C_1 = \{x_1\}, \dots, C_n = \{x_n\}$

Repeat

 Finde zwei Cluster C_i und C_j mit kleinstem Abstand
 Vereinige C_i und C_j

Until Abbruchbedingung erreicht

Return(Baum mit Clustern)

Als Abbruchbedingung könnte zum Beispiel eine gewünschte Zahl von Clustern oder ein maximaler Abstand der Cluster gewählt werden. In Abb. 8.34 ist dieses Verfahren schematisch als Binärbaum dargestellt, bei dem von unten nach oben in jedem Schritt, das heißt auf jeder Ebene, zwei Unteräume verbunden werden. Auf der obersten Ebene sind alle Punkte zu einem großen Cluster vereinigt.

Unklar ist bislang noch, wie die Abstände der Cluster berechnet werden. Wir haben zwar im vorigen Abschnitt verschiedene Abstandsmaße für Punkte definiert, diese sind aber auf Cluster nicht anwendbar. Ein naheliegendes und auch oft verwendetes Maß ist der Abstand der zwei nächstliegenden Punkte aus den beiden Clustern C_i und C_j , also

$$d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y).$$

Damit erhält man den **Nearest Neighbour-Algorithmus**, dessen Arbeitsweise in Abb. 8.35 dargestellt ist.¹⁰ Man erkennt, dass dieser Algorithmus einen minimal aufspannenden Baum¹¹ erzeugt. Außerdem zeigt das Beispiel, dass die beiden beschriebenen Algorithmen ganz unterschiedliche Cluster erzeugen. Daraus lernen wir, dass bei Graphen mit Clustern,

¹⁰ Der Nearest Neighbour-Algorithmus ist nicht zu verwechseln mit der Nearest Neighbour-Methode zur Klassifikation aus Abschn. 8.3.

¹¹ Ein minimal aufspannender Baum ist ein zyklenfreier ungerichteter Graph mit minimaler Summe der Kantenlängen.

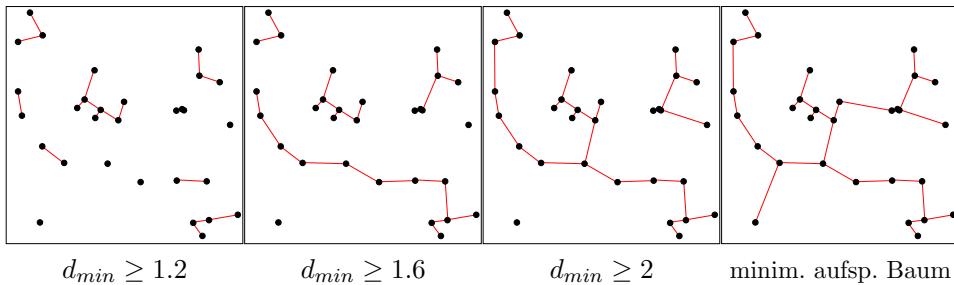


Abb. 8.35 Der Nearest Neighbour-Algorithmus, angewandt auf die Daten aus Abb. 8.33 auf verschiedenen Ebenen mit 12, 6, 3, 1 Clustern

die nicht klar getrennt sind, das Ergebnis stark vom Algorithmus oder vom gewählten Abstandsmaß abhängt.

Für eine effiziente Implementierung dieses Algorithmus erstellt man zu Beginn eine Adjazenzmatrix, in der die Abstände zwischen allen Punkten gespeichert werden, was $O(n^2)$ Zeit und Speicherplatz erfordert. Falls die Zahl der Cluster nicht nach unten beschränkt ist, wird die Schleife $n - 1$ mal durchlaufen und als asymptotische Rechenzeit ergibt sich $O(n^3)$.

Zur Berechnung des Abstands von zwei Clustern kann man auch den Abstand der beiden entferntesten Punkte

$$d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y).$$

verwenden und erhält den **Farthest Neighbour-Algorithmus**. Alternativ wird auch oft der Abstand der Clustermittelpunkte $d_\mu(C_i, C_j) = d(\mu_i, \mu_j)$ benutzt. Neben den hier vorgestellten Clustering-Algorithmen gibt es noch viele weitere, für deren Studium wir auf [DHS01] verweisen.

8.10 Data Mining in der Praxis

Alle bisher vorgestellten Lernverfahren können als Werkzeug zum Data Mining verwendet werden. Für den Benutzer ist es aber teilweise recht mühsam, sich je nach Anwendung in immer wieder neue Software-Werkzeuge einzuarbeiten und außerdem die zu analysierenden Daten in das jeweils passende Format zu bringen.

Dieses Problem adressieren eine ganze Reihe von Data Mining-Systemen. Die meisten dieser Systeme bieten eine komfortable grafische Benutzeroberfläche mit diversen Werkzeugen zur Visualisierung der Daten, zur Vorverarbeitung, wie zum Beispiel die Manipulation von fehlenden Werten, und zur Analyse. Bei der Analyse werden unter anderem die hier vorgestellten Lernverfahren eingesetzt.

Eine besondere Erwähnung verdient die in [WF01] vorgestellte umfassende Open-Source Java-Programmbibliothek WEKA, welche eine große Anzahl an Algorithmen zum Data Mining anbietet und die Entwicklung neuer Verfahren vereinfacht.

Das frei verfügbare System KNIME, das wir im folgenden Abschnitt kurz vorstellen werden, bietet eine komfortable Benutzeroberfläche und alle Arten der oben erwähnten Werkzeuge. KNIME verwendet unter anderem WEKA-Module. Außerdem bietet es eine einfache Möglichkeit, mittels eines graphischen Editors den Datenfluss der ausgewählten Visualisierungs-, Vorverarbeitungs- und Analysewerkzeuge zu kontrollieren. Eine ganz ähnliche Funktionalität bietet eine mittlerweile große Zahl anderer Systeme, wie zum Beispiel das Open-Source-Projekt RapidMiner (<http://www.rapidminer.com>), das von SPSS vertriebene System Clementine (<http://www.spss.com/clementine>) oder das KXEN Analytic Framework (<http://www.lxen.com>).

8.10.1 Das Data Mining Werkzeug KNIME

Wir zeigen nun anhand der LEXMED-Daten, wie man mit dem Data Mining-Werkzeug KNIME (Konstanz Information Miner, <http://www.knime.org>) Wissen aus Daten extrahieren kann. Zuerst generieren wir einen Entscheidungsbaum wie in Abb. 8.36 dargestellt. Nach der Definition eines neuen Projekts wird ein Workflow grafisch aufgebaut. Dazu werden einfach die entsprechenden Werkzeuge aus dem Node-Repository mit der Maus in das Hauptfenster mit dem Workflow gezogen.

Mit den beiden File Reader-Knoten lassen sich die Trainings- und die Testdaten aus der C4.5-Datei problemlos einlesen. Ganz einfach lassen sich diese Knoten aber auch für andere Dateiformate konfigurieren. Die liegende Ampel unter dem Knoten zeigt den Status (unfertig, konfiguriert, ausgeführt) an. Dann wird aus der WEKA-Bibliothek [WF01] der Knoten J48 gewählt, welcher eine Java-Implementierung von C4.5 enthält. Die Konfiguration ist auch hier ganz einfach. Nun wird noch ein Predictor-Knoten gewählt, der den generierten Baum auf die Testdaten anwendet. Er fügt zur Testdatentabelle eine neue Spalte „Prediction“ mit der durch den Baum erzeugten Klassifikation hinzu. Daraus berechnet dann der Scorer-Knoten die in der Abbildung dargestellte Konfusionsmatrix, welche in der Diagonalen die Anzahl von korrekt klassifizierten Fällen für beide Klassen angibt und außerdem die Zahl der falsch positiven und falsch negativen Daten.

Ist der Ablauf komplett aufgebaut und alle Knoten konfiguriert, so kann ein beliebiger Knoten ausgeführt werden. Er sorgt automatisch dafür, dass Vorgängerknoten ausgeführt werden, falls nötig. Der J48-Knoten erzeugt den in der Abbildung rechts dargestellten „View“ des Entscheidungsbaums. Dieser Baum stimmt übrigens exakt mit dem von C4.5 in Abschn. 8.4.5 erzeugten Baum überein, wobei hier der Knoten $T \text{Rekt} \leq 378$ eingeklappt ist.

Zum Vergleich ist in Abb. 8.37 ein Projekt zum Lernen eines mehrlagigen Perzeptrons (siehe Abschn. 9.5) dargestellt. Dieses arbeitet ähnlich dem schon vorgestellten linearen Perzeptron, kann aber auch nichtlinear separable Klassen trennen. Hier ist der Ablauf etwas

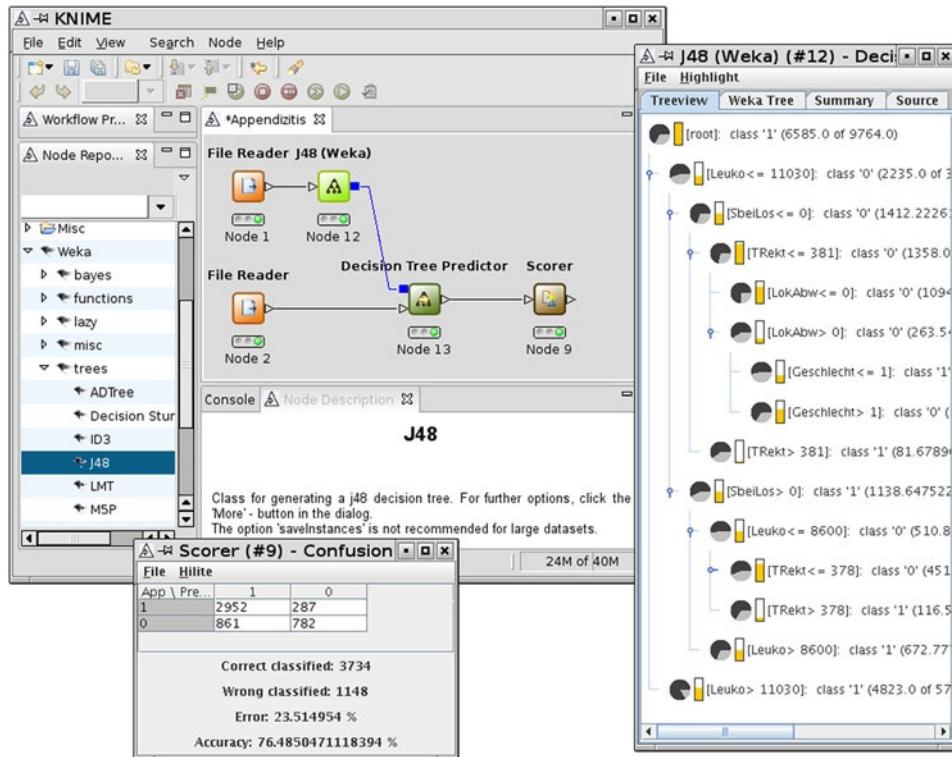


Abb. 8.36 Die KNIME-Benutzeroberfläche mit zwei zusätzlichen Views, die den Entscheidungsbaum und die Konfusionsmatrix darstellen

komplexer. Es wird zur Vorverarbeitung für jede der beiden Dateien noch ein Knoten für die Behandlung von fehlenden Werten benötigt. Wir stellen ein, dass entsprechende Zeilen gelöscht werden sollen. Da neuronale Netze nicht mit beliebigen Werten umgehen können, werden die Werte von allen Variablen mittels der „Normalizer“-Knoten noch linear in das Intervall $[0, 1]$ skaliert.

Nach Anwendung des RProp-Lerners, einer Verbesserung von Backpropagation (siehe Abschn. 9.5), kann man in der dargestellten Lernkurve den zeitlichen Verlauf des Approximationsfehlers analysieren. In der Konfusionsmatrix gibt der Scorer die Analyse auf den Testdaten aus. Der „CSV-Writer“-Knoten rechts unten dient zum Export der Ergebnis-Daten, die dann noch extern zur Erzeugung der in Abb. 7.10 dargestellten ROC-Kurve verwendet werden, wofür es leider (noch) kein passendes KNIME-Werkzeug gibt.

Zusammenfassend lässt sich über KNIME (und ähnliche Werkzeuge) folgendes sagen: Für Projekte mit nicht allzu exotischen Anforderungen an die Datenanalyse lohnt es sich, mit solch einer mächtigen Werkbank zur Analyse der Daten zu arbeiten. Schon bei der Vorverarbeitung der Daten spart der Anwender viel Zeit. Er hat eine große Aus-

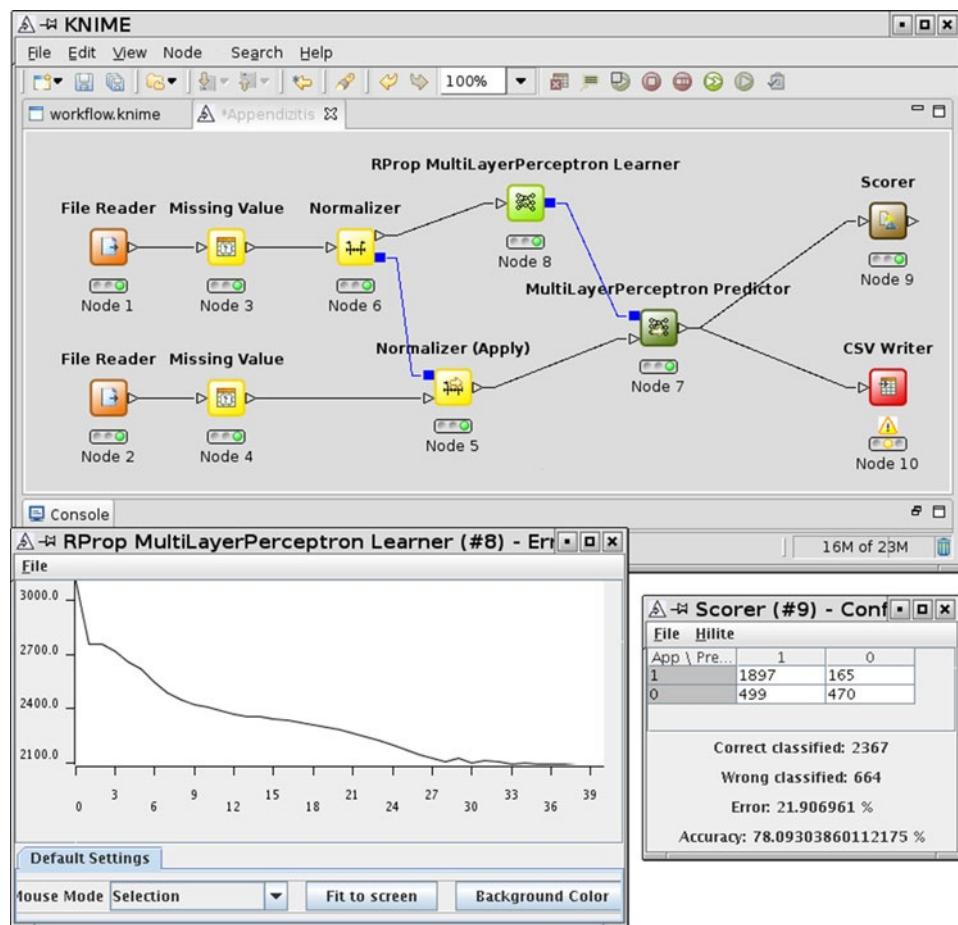


Abb. 8.37 Die KNIME-Benutzeroberfläche mit dem Workflow-Fenster, der Lernkurve und der Konfusionsmatrix

wahl an leicht verwendbaren „Mining-Werkzeugen“, unter anderem zum Beispiel auch Nearest-Neighbour-Klassifizierer, einfache Bayes-Netz-Lernverfahren sowie das k -Means Clustering-Verfahren (Abschn. 8.9.2). Auch die Auswertung der Ergebnisse, zum Beispiel mittels Kreuzvalidierung, lässt sich einfach durchführen. Bleibt nur noch zu erwähnen, dass es neben den gezeigten viele weitere Werkzeuge zur Visualisierung der Daten gibt. Außerdem bieten die Entwickler von KNIME eine Erweiterung des Systems an, mit der der Anwender selbst in Java oder Python eigene Werkzeuge programmieren kann.

Allerdings sollte auch erwähnt werden, dass der Anwender eines derartigen Data Mining-Systems solides Vorwissen über die verwendeten Techniken des Data Mining und des maschinellen Lernens mitbringen sollte. Die Software alleine analysiert keine Daten.

Aber in der Hand eines Fachmanns wird sie zum starken Werkzeug für die Extraktion von Wissen aus Daten. Gerade für den Einsteiger in das faszinierende Gebiet des maschinellen Lernens bietet solch ein System eine ideale und einfache Gelegenheit, das Wissen praktisch zu erproben und die verschiedenen Verfahren zu vergleichen. Der Leser möge dies verifizieren anhand von Aufgabe 8.22.

8.11 Zusammenfassung

Ausführlich behandelt wurden einige Verfahren wie etwa das Lernen von Entscheidungsbäumen, Bayes-Netzen und die Nearest Neighbour-Methoden aus dem heute etablierten Gebiet des Lernens mit Lehrer. Diese Verfahren sind stabil und effizient einsetzbar in den verschiedensten Anwendungen und gehören daher heute zum Standardrepertoire in KI und Data Mining. Ähnliches gilt für die ohne „Lehrer“ arbeitenden Clusteringverfahren, die zum Beispiel in Suchmaschinen Verwendung finden.

Lernen durch Verstärkung ist ein anderes Teilgebiet des Maschinellen Lernens, das ohne einen expliziten Lehrer auskommt. Im Gegensatz zum Lernen mit Lehrer, bei dem der Lerner die korrekten Aktionen oder Antworten als Labels in den Trainingsdaten vorgegeben bekommt, erhält der lernende Agent beim Lernen durch Verstärkung nur ab und zu ein positives oder negatives Feedback von der Umgebung (siehe Kap. 10). Nicht ganz so schwierig ist die Aufgabe beim Semi-Supervised Learning, einem jungen Teilgebiet des maschinellen Lernens bei dem nur einige wenige der vielen Trainingsdaten ein Label besitzen.

Das Lernen mit Lehrer ist heute eine etablierte Ingenieursdisziplin mit vielen erfolgreichen Anwendungen. Wenn Daten mit stetigen Labels gegeben sind, stehen viele verschiedene Funktionsapproximationsalgorithmen aus Mathematik oder Informatik zur Verfügung. In Kap. 9 werden wir verschiedene Arten von Neuronalen Netzen vorstellen, aber auch die Methode der kleinsten Quadrate und Support Vector Maschinen, die alle Funktionsapproximatoren sind. Von besonderem Interesse sind Gauß'sche Prozesse, weil sie sehr universell und einfach anwendbar sind und dem Nutzer auch noch eine Schätzung der Unsicherheit der Antworten liefern [RW06].

Die Taxonomie in Tab. 8.7 soll einen Überblick über die wichtigsten Lernverfahren und deren Klassifizierung zu geben.

Das zum Lernen mit Lehrer Gesagte gilt aber nur, wenn mit einer festen Menge von bekannten Merkmalen gearbeitet wird. Ein interessantes, noch weitgehend offenes Gebiet, in dem intensiv geforscht wird, ist die automatische Merkmalsextraktion (engl. feature selection). Beim Lernen von Entscheidungsbäumen wurde ein Ansatz hierzu gezeigt. Das vorgestellte Verfahren der Berechnung des Informationsgewinns von Merkmalen sortiert die Merkmale nach ihrer Relevanz und verwendet nur diejenigen, welche die Klassifikationsgüte verbessern. Mit einer derartigen Methode ist es also möglich, aus einer eventuell großen Grundmenge von Attributen die relevanten automatisch auszuwählen. Diese Grundmenge muss aber manuell ausgewählt werden.

Tab. 8.7 Taxonomie der maschinellen Lernverfahren**Lernen mit Lehrer**

- faules Lernen (lazy learning)
 - k-Nearest-Neighbour-Methode (Klass. + Approx.)
 - lokal gewichtete Regression (Approx.)
 - Fallbasiertes Lernen (Klass. + Approx.)
 - One-Class nearest Neighbour (Klass. + Approx.)
- eifriges Lernen (eager learning)
 - Induktion von Entscheidungsbäumen (Klass.)
 - Lernen von Bayes-Netzen (Klass. + Approx.)
 - neuronale Netze (Klass. + Approx.)
 - Support-Vektor-Maschinen (Klass. + Approx.)
 - Gauß'sche Prozesse (Klass. + Approx.)
 - Pseudoinverse, Wavelets, Radial Basis Funktionen, ...
 - One-Class Support Vector Maschinen (Klass. + Approx.)

Lernen ohne Lehrer (Clustering)

- Nearest Neighbour-Algorithmus
- Farthest Neighbour-Algorithmus
- k-Means
- neuronale Netze

Lernen durch Verstärkung

- Wert-Iteration
- Q-Lernen
- TD-Lernen
- Policy-Gradient-Methoden
- neuronale Netze

Noch ungeklärt und auch nicht klar definiert ist die Frage, ob und wie die Maschine neue Merkmale finden kann. Stellen wir uns einen Roboter vor, der Äpfel pflücken soll. Dazu soll er lernen, reife Äpfel von unreifen Äpfeln und anderen Objekten zu unterscheiden. Klassisch wird man aus einem Videobild mittels Bildverarbeitung bestimmte Attribute wie zum Beispiel Farbe und Form von Pixelbereichen ermitteln und dann ein Lernverfahren anhand von manuell klassifizierten Bildern trainieren. Denkbar ist es auch, dass zum Beispiel ein neuronales Netz direkt mit allen Pixeln des Bildes als Eingabe trainiert wird, was bei hoher Auflösung allerdings mit großen Rechenzeitproblemen verbunden ist. Interessant sind hier Ansätze, die automatisch Vorschläge für relevante Merkmale machen.

Einen Ansatz zur Merkmalsextraktion liefert das Clustering. Vor dem Training der Apfelerkennungsmaschine lässt man ein Clustering auf den Daten laufen. Als Eingabe für das Lernen (mit Lehrer) der Klassen *Apfel* oder *nicht Apfel* werden dann nicht mehr alle Pixel verwendet, sondern nur noch die Zugehörigkeit zu den beim Clustering gefundenen Klassen, eventuell zusammen mit anderen Attributen. Das Clustering jedenfalls kann zu einem

automatischen kreativen „Entdecken“ von Merkmalen verwendet werden. Es ist allerdings nicht sicher, dass die gefundenen Merkmale relevant sind.

Noch schwieriger ist folgendes Problem: Angenommen, die zur Apfelerkennung verwendete Videokamera liefert nur Schwarz-Weiß-Bilder. Dann wird die Aufgabe nicht mehr gut gelöst werden können. Schön wäre es, wenn dann die Maschine von sich aus kreativ würde, zum Beispiel mit dem Vorschlag, die Kamera durch eine Farbkamera zu ersetzen. Dies wäre aber heute noch etwas zu viel verlangt.

Neben Spezialwerken über alle Teilgebiete des maschinellen Lernens gibt es die sehr guten Lehrbücher [Mit97, Bis06, DHS01, HTF09, Alp04]. Für aktuelle Forschungsergebnisse empfiehlt sich ein Blick in das frei verfügbare Journal of Machine Learning Research (<http://jmlr.csail.mit.edu>), das Machine Learning Journal sowie in die Proceedings der International Conference on Machine Learning (ICML). Für jeden Entwickler von Lernverfahren interessant ist das Machine Learning Repository [DNM98] der Universität Irvine (UCI) mit einer großen Sammlung von Trainings- und Testdaten für Lernverfahren und Data Mining-Werkzeuge.

Ein hervorragendes Verzeichnis frei verfügbarer Software für maschinelles Lernen ist MLOSS (Machine Learning Open Source Software, <http://www.mloss.org>).

8.12 Übungen

Einführung

Aufgabe 8.1

- Spezifizieren Sie die Aufgabe eines Agenten, der anhand gemessener Werte für Temperatur, Luftdruck und Luftfeuchtigkeit an drei aufeinander folgenden Tagen eine Prognose des Wetters für den nächsten Tag erstellt. Das Wetter soll eingeteilt werden in die drei Klassen sonnig, bewölkt und regnerisch.
- Beschreiben Sie nun die Struktur einer Datei mit Trainingsdaten für diesen Agenten.

Aufgabe 8.2

Zeigen Sie, dass die Korrelationsmatrix symmetrisch ist und dass alle Diagonalelemente gleich 1 sind.

Das Perzeptron

Aufgabe 8.3

Wenden Sie die Perzeptron-Lernregel auf die Mengen

$$M_+ = \{(0, 1, 8), (2, 0, 6)\} \quad \text{und} \quad M_- = \{(-1, 2, 1, 4), (0, 4, -1)\}$$

aus Beispiel 8.2 an und geben Sie die Folge der Werte für den Gewichtsvektor an.

Aufgabe 8.4 Gegeben sei die folgende Tabelle mit Trainingsdaten:

Nr.	x_1	x_2	Klasse
1	6	1	0
2	7	3	0
3	8	2	0
4	9	0	0
5	8	4	1
6	8	6	1
7	9	2	1
8	9	5	1

- a) Zeigen Sie anhand einer Zeichnung, dass die Daten linear separabel sind.
- b) Bestimmen Sie manuell die Gewichte w_1 und w_2 sowie die Schwelle Θ eines Perzeptrons (mit Schwelle), das die Daten korrekt klassifiziert.
- c) Programmieren Sie die Perzeptron-Lernregel und wenden Sie Ihr Programm auf obige Tabelle an. Vergleichen Sie die gefundenen Gewichte mit den von Ihnen berechneten.

Aufgabe 8.5

- a) Geben Sie eine anschauliche Interpretation der in Abschn. 8.2.2 beschriebenen heuristischen Initialisierung

$$\mathbf{w}_0 = \sum_{x_i \in M_+} x_i - \sum_{x_i \in M_-} x_i,$$

des Gewichtsvektors beim Perzeptron.

- b) Geben Sie ein Beispiel für eine linear separable Datenmenge an, für die diese Heuristik keine Trenngerade darstellt.

Nearest-Neighbour-Methode

Aufgabe 8.6



- a) Zeichnen Sie für obige Punktmenge das Voronoi-Diagramm.
- b) Zeichnen Sie dann die Klassentrennlinie(n).

Aufgabe 8.7 Gegeben sei die Tabelle mit Trainingsdaten aus Aufgabe 8.4. Zur Bestimmung des Abstandes d von zwei Datenpunkten $\mathbf{a} = (a_1, a_2)$ und $\mathbf{b} = (b_1, b_2)$ verwenden Sie im Folgenden den Manhattan-Abstand $d(\mathbf{a}, \mathbf{b})$, der definiert ist als $d(\mathbf{a}, \mathbf{b}) = |a_1 - b_1| + |a_2 - b_2|$.

- Klassifizieren Sie mit der Nearest Neighbour-Methode den Vektor $\mathbf{v} = (8, 3, 5)$.
- Klassifizieren Sie mit der k -Nearest Neighbour-Methode den Vektor $\mathbf{v} = (8, 3, 5)$ für $k = 2, 3, 5$.

Aufgabe 8.8 *

- Zeigen Sie, dass es bei Verwendung der Euklidischen Norm sinnvoll ist, wie in (8.3) gefordert, die k nächsten Nachbarn entsprechend dem Kehrwert des Abstandsquadrats zu gewichten.
- Warum wäre eine Gewichtung mittels $w'_i = \frac{1}{1+\alpha d(x, x_i)}$ weniger sinnvoll?

Aufgabe 8.9

- Programmieren Sie die k -Nearest-Neighbour-Methode zur Klassifikation.
- Wenden Sie Ihr Programm für $k = 1$ (d. h. ein nächster Nachbar) auf die Lexmed-Daten aus <http://www.hs-weingarten.de/~ertel/kibuch/uebungen/> an. Verwenden Sie zum Trainieren die Datei app1.data und zum Testen app1.test. Vergessen Sie nicht, vor dem Training alle Merkmale zu normalisieren.
- Finden Sie nun mittels Leave-One-Out-Kreuzvalidierung auf der gesamten Datenmenge app1.data ∪ app1.test die optimale Zahl nächster Nachbarn k und bestimmen Sie den Klassifikationsfehler.
- Wiederholen Sie die Kreuzvalidierung mit nicht normalisierten Daten.
- Vergleichen Sie Ihre Ergebnisse mit den in Abb. 9.14 dargestellten für die Methode der kleinsten Quadrate und RProp.

Entscheidungsbäume

Aufgabe 8.10 Zeigen Sie, dass die Definition $0 \log_2 0 := 0$ sinnvoll ist, d. h. dass die Funktion $f(x) = x \log_2 x$ dadurch im Nullpunkt stetig ergänzt wird.

Aufgabe 8.11 Bestimmen Sie die Entropie für folgende Verteilungen

- $(1, 0, 0, 0, 0)$
- $(\frac{1}{2}, \frac{1}{2}, 0, 0, 0)$
- $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0, 0)$
- $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16})$
- $(\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5})$
- $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \dots)$

Aufgabe 8.12

- a) Zeigen Sie, dass sich die zwei verschiedenen Definitionen der Entropie aus (7.9) und Definition 8.4 nur um einen konstanten Faktor unterscheiden, das heißt, dass

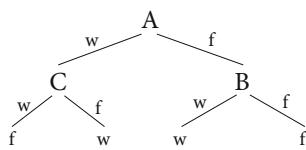
$$\sum_{i=1}^n p_i \log_2 p_i = c \sum_{i=1}^n p_i \ln p_i$$

und geben Sie die Konstante c an.

- b) Zeigen Sie, dass es bei der MaxEnt-Methode und beim Lernen von Entscheidungsbäumen keine Rolle spielt, welche der beiden Formeln man verwendet.

Aufgabe 8.13 Entwickeln Sie einen Entscheidungsbaum für die Datenmenge D aus Aufgabe 8.4.

- a) Behandeln Sie beide Attribute als diskret.
- b) Behandeln Sie nun Attribut x_2 als stetig und x_1 diskret.
- c) Lassen Sie C4.5 mit beiden Varianten einen Baum erzeugen. Verwenden Sie hierbei als Parameter `-m 1 -t 10`, um verschiedene Vorschläge zu erhalten.

Aufgabe 8.14 Gegeben sei der folgende Entscheidungsbaum und je eine Tabelle mit Trainings- und Testdaten.

Trainingsdaten			
A	B	C	Klasse
w	w	f	w
w	f	f	w
w	w	w	f
f	f	w	f
f	f	f	f
f	w	w	w

Testdaten			
A	B	C	Klasse
w	w	f	w
w	f	f	w
f	w	f	f
f	f	w	f

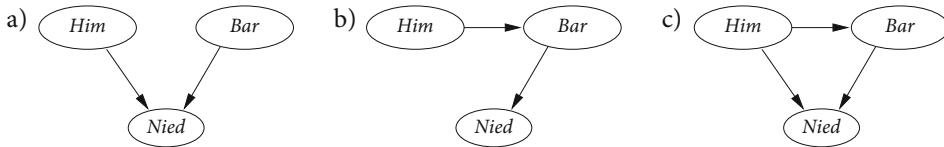
- a) Geben Sie die Korrektheit des Baumes auf den Trainings- und Testdaten an.
- b) Geben Sie eine zu dem Baum äquivalente aussagenlogische Formel an.
- c) Führen Sie auf dem Baum das Pruning durch, zeichnen Sie den resultierenden Baum und geben Sie seine Korrektheit auf den Trainings- und Testdaten an.

Aufgabe 8.15 *

- a) Der Algorithmus zum Erzeugen von Entscheidungsbäumen (Abb. 8.25) schließt bei der Bestimmung des aktuellen Attributs die weiter oben im Baum schon verwendeten Attribute nicht aus. Trotzdem kommt ein diskretes Attribut in einem Pfad höchstens einmal vor. Warum?
- b) Warum können jedoch die stetigen Attribute mehrfach vorkommen?

Lernen von Bayes-Netzen

Aufgabe 8.16 Verwenden Sie die in Aufgabe 7.3 gegebene Verteilung und bestimmen Sie die CPTs für die drei Bayes-Netze:



- d) Bestimmen Sie für die beiden Netze aus a) und b) die Verteilung und vergleichen diese mit der Original-Verteilung. Welches Netz ist das „bessere“?
 e) Bestimmen Sie nun die Verteilung für das Netz c). Was fällt Ihnen auf? Begründung!

Aufgabe 8.17 *** Zeigen Sie, dass für binäre Variablen S_1, \dots, S_n und binäre Klassenvariable K ein linearer Score der Form

$$\text{Entscheidung} = \begin{cases} \text{positiv} & \text{falls } w_1 S_1 + \dots + w_n S_n > \Theta \\ \text{negativ} & \text{sonst} \end{cases}$$

bezüglich der Ausdrucksstärke äquivalent zum Perzeptron und zum Naive-Bayes-Klassifizierer ist, die beide nach der Formel

$$\text{Entscheidung} = \begin{cases} \text{positiv} & \text{falls } P(K | S_1, \dots, S_n) > 1/2 \\ \text{negativ} & \text{sonst} \end{cases}$$

entscheiden.

Aufgabe 8.18 Bei der Implementierung der Textklassifikation mit Naive-Bayes kann es schnell zu einem Exponentenunterlauf kommen, denn die in dem Produkt aus (8.10) vorkommenden Faktoren $P(w_i | K)$ sind typischerweise allesamt sehr klein, was zu extrem kleinen Ergebnissen führen kann. Wie kann man dieses Problem entschärfen?

Aufgabe 8.19 $\Rightarrow \ast$ Schreiben Sie ein Programm zur Naive-Bayes-Textanalyse. Trainieren und testen Sie dieses dann auf den Textbenchmarks. Verwenden Sie dazu ein Werkzeug Ihrer Wahl. Das Zählen der Häufigkeiten von Wörtern im Text kann unter Linux durch den Befehl

```
cat <datei> | tr -d "[:punct:]" | tr -s "[:space:]" "\n" | sort | uniq -ci
```

einfach erledigt werden.

Besorgen Sie sich dazu in der UCI Machine Learning Benchmark-Sammlung die 20-Newsgroup-Daten von Tom Mitchell [DNM98]. Dort finden Sie auch einen Verweis auf ein Naive-Bayes-Programm zur Textklassifikation von Mitchell.

Clustering

Aufgabe 8.20 Zeigen Sie, dass bei Algorithmen, die nur Abstände vergleichen, die Anwendung einer streng monoton wachsenden Funktion f auf den Abstand keinen Unterschied macht. Es ist also zu zeigen, dass der Abstand $d_1(x, y)$ und der Abstand $d_2(x, y) := f(d_1(x, y))$ zum gleichen Ergebnis bezüglich der Größer-Relation führen.

Aufgabe 8.21 Bestimmen Sie die Abstände d_s (Skalarprodukt) der folgenden Texte untereinander.

- x_1 : Die eigentliche Anwendung von Naive-Bayes auf die Textanalyse wollen wir an einem kurzen Beispieltext von Alan Turing aus [Tur50] vorstellen:
- x_2 : Wir dürfen hoffen, dass Maschinen vielleicht einmal auf allen rein intellektuellen Gebieten mit dem Menschen konkurrieren. Aber mit welchem sollte man am besten beginnen?
- x_3 : Wiederum weiß ich nicht, welches die richtige Antwort ist, aber ich meine, dass man beide Ansätze erproben sollte.

Data Mining

Aufgabe 8.22 Benutzen Sie KNIME (<http://www.knime.de>) und

- a) Laden Sie die Beispiel-Datei mit den Iris-Daten aus dem KNIME-Verzeichnis und experimentieren Sie mit den verschiedenen Daten-Darstellungen, insbesondere mit Streu-Diagrammen.
- b) Trainieren Sie nun zuerst einen Entscheidungsbaum für die drei Klassen und dann ein RProp-Netzwerk.
- c) Laden Sie die Appendizitis-Daten auf der Website zum Buch. Vergleichen Sie die Klassifikationsgüte der k -Nearest-Neighbour-Methode mit der eines RProp-Netzwerks. Optimieren Sie sowohl k als auch die Zahl der verdeckten Neuronen des RProp-Netzes.
- d) Besorgen Sie sich auf der UCI-Datensammlung für Data Mining unter <http://kdd.ics.uci.edu> oder für Machine Learning unter <http://mlearn.ics.uci.edu/MLRepository.html> eine Datenmenge Ihrer Wahl und experimentieren Sie damit.

Neuronale Netze sind Netzwerke aus Nervenzellen im Gehirn von Menschen und Tieren. Etwa 10 bis 100 Milliarden Nervenzellen besitzt das menschliche Gehirn. Der komplexen Verschaltung und der Adaptivität verdanken wir Menschen unsere Intelligenz und unsere Fähigkeit, verschiedenste motorische und intellektuelle Fähigkeiten zu lernen und uns an variable Umweltbedingungen anzupassen. Schon seit vielen Jahrhunderten versuchen Biologen, Psychologen und Mediziner die Funktionsweise von Gehirnen zu verstehen. Um das Jahr 1900 wuchs die revolutionäre Erkenntnis, dass eben diese winzigen physikalischen Bausteine des Gehirns, die Nervenzellen und deren komplexe Verschaltung, für Wahrnehmung, Assoziationen, Gedanken, Bewusstsein und die Lernfähigkeit verantwortlich sind.

Den großen Schritt hin zu einer KI der neuronalen Netze wagten dann 1943 McCulloch und Pitts in einem Artikel mit dem Titel „A logical calculus of the ideas immanent in nervous activity“ [AR88]. Sie waren die ersten, die ein mathematisches Modell des Neurons als grundlegendes Schaltelement für Gehirne vorstellten. Dieser Artikel legte die Basis für den Bau von künstlichen neuronalen Netzen und damit für dieses ganz wichtige Teilgebiet der KI.

Man kann das Gebiet der Modellierung und Simulation von neuronalen Netzen auch als den Bionik-Zweig innerhalb der KI verstehen.¹ In fast allen Gebieten der KI wird versucht, kognitive Prozesse nachzubilden, etwa in der Logik oder beim probabilistischen Schließen. Die verwendeten Werkzeuge zur Modellierung – nämlich Mathematik, Programmiersprachen und Digitalrechner – haben aber wenig Ähnlichkeit mit einem menschlichen Gehirn. Bei den künstlichen neuronalen Netzen ist der Zugang ein anderer. Ausgehend von dem Wissen über die Funktion natürlicher neuronaler Netze versucht man, diese zu modellieren, zu simulieren oder sogar in Hardware nachzubauen. Hierbei stellt sich jedem Forscher die faszinierende und spannende Herausforderung, die Ergebnisse mit der Leistungsfähigkeit von uns Menschen zu vergleichen.

¹ Die Bionik beschäftigt sich mit der Entschlüsselung von „Erfindungen der belebten Natur“ und ihrer innovativen Umsetzung in der Technik [Wik13].

Wir werden in diesem Kapitel versuchen, den durch die Geschichte vorgegebenen Weg zu skizzieren, indem wir, ausgehend von den wichtigsten biologischen Erkenntnissen, das Modell des Neurons und dessen Vernetzung definieren. Dann werden wir mit dem Hopfield-Modell, zwei einfachen Assoziativspeichermodellen und dem für die Praxis überaus wichtigen Backpropagation-Algorithmus einige wichtige und grundlegende Modelle vorstellen.

9.1 Von der Biologie zur Simulation

Jedes der etwa 100 Milliarden Neuronen in einem menschlichen Gehirn hat, wie in Abb. 9.1 vereinfacht dargestellt, folgende Struktur und Funktion. Neben dem Zellkörper besitzt das Neuron ein Axon, welches im Gehirn über die Dendriten lokale Verbindungen zu anderen Neuronen herstellen kann. Das Axon kann aber auch in Form einer Nervenfaser durch den Körper bis etwa einen Meter lang werden.

Der Zellkörper des Neurons stellt einen Speicher für kleine elektrische Spannungen, vergleichbar mit einem Kondensator oder Akku, dar. Dieser Speicher wird aufgeladen durch eingehende Spannungsimpulse von anderen Neuronen. Je mehr Spannungsimpulse ankommen, umso höher wird die Spannung. Überschreitet die Spannung einen gewissen Schwellwert, so feuert das Neuron. Das heißt, es entlädt seinen Speicher, indem es einen Spannungsimpuls (engl. spike) über das Axon und die Synapsen losschickt. Der Strom teilt sich auf und erreicht über die Synapsen schließlich viele andere Neuronen, in denen der gleiche Prozess abläuft.

Nun stellt sich die Frage nach der Struktur des Neuronennetzes. Im Gehirn ist jedes der etwa 10^{11} Neuronen mit etwa 1000 bis 10.000 anderen Neuronen verbunden, was insgesamt über 10^{14} Verbindungen ergibt. Bedenkt man dann noch, dass diese riesige Zahl von extrem dünnen Verbindungen in einem dreidimensionalen weichen Gewebe eingelagert sind und dass Versuche an menschlichen Gehirnen nicht einfach durchzuführen sind, so wird klar, weshalb wir heute den Schaltplan des Gehirns noch nicht im Detail kennen. Vermutlich werden wir, schon allein aufgrund der immensen Größe, niemals in der Lage sein, den Schaltplan unseres Gehirns komplett zu verstehen.

Einen genauen Schaltplan eines Gehirns zu erstellen, ist aber aus heutiger Sicht gar nicht mehr erstrebenswert, denn die Struktur des Gehirns ist adaptiv. Sie verändert sich laufend und passt sich entsprechend den Aktivitäten des Individuums an die Umwelteinflüsse an. Die zentrale Rolle hierbei spielen die Synapsen, welche die Verbindung zwischen den Neuronen herstellen. An der Verbindungsstelle zwischen zwei Neuronen treffen quasi zwei Kabel aufeinander. Nun sind die beiden Leitungen aber nicht perfekt leitend verbunden, sondern es gibt einen kleinen Spalt, den die Elektronen nicht direkt überwinden können. Dieser Spalt ist gefüllt mit chemischen Substanzen, den so genannten Neurotransmittern. Durch eine anliegende Spannung können diese ionisiert werden und dann Ladung über den Spalt transportieren. Die Leitfähigkeit dieses Spalts hängt von vielen Parametern ab. Zum Beispiel von der Konzentration und der chemischen Zusammensetzung der Neurotransmitter. Es ist einleuchtend, dass die Funktion eines Gehirns sehr empfindlich reagiert auf Verän-

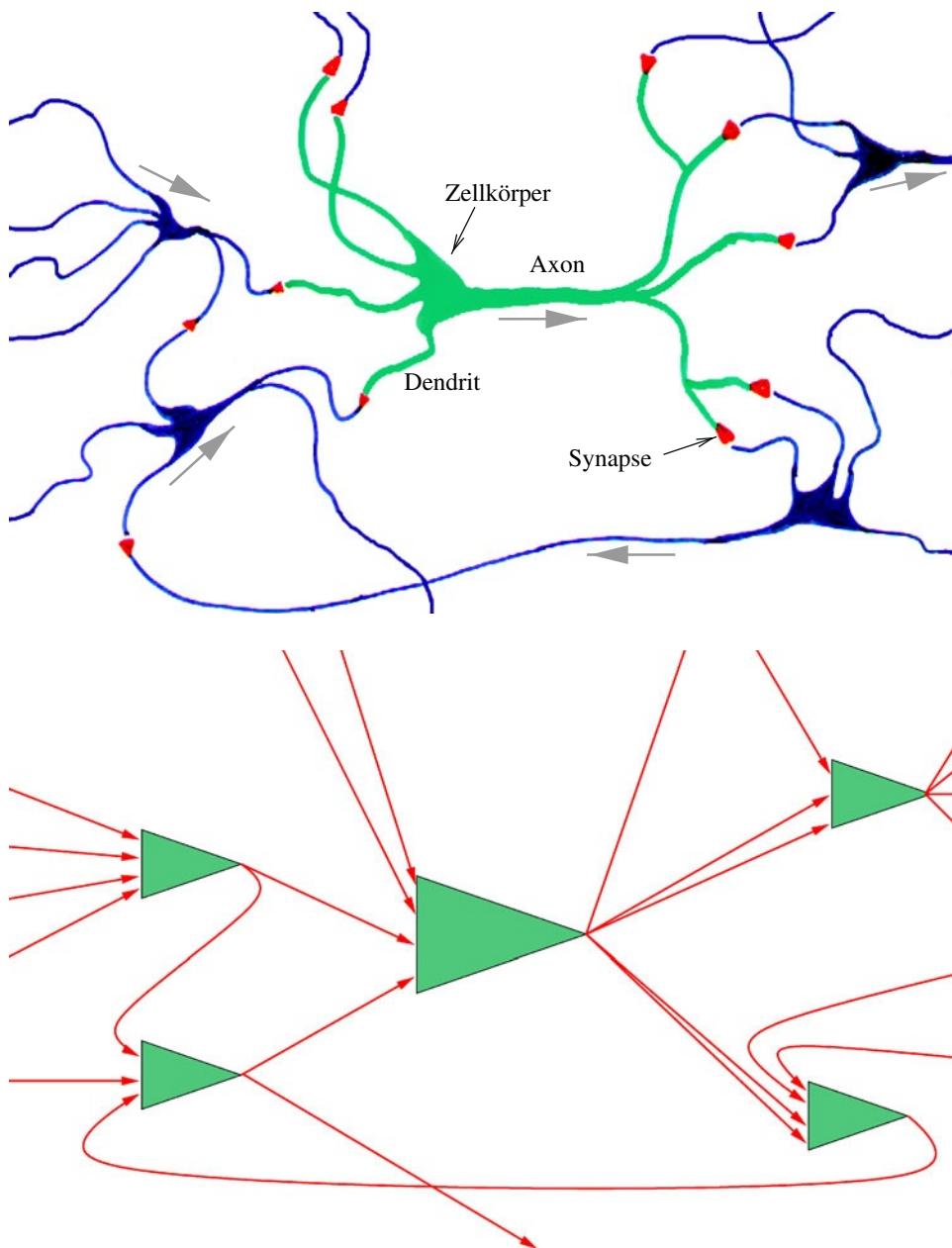


Abb. 9.1 Zwei Stufen der Modellierung eines neuronalen Netzes. Oben ein biologisches Modell und unten ein formales Modell mit Neuronen und gerichteten Verbindungen zwischen ihnen

derungen dieser synaptischen Verbindungen, zum Beispiel durch den Einfluss von Alkohol oder anderen Drogen.

Wie funktioniert nun das Lernen in solch einem neuronalen Netz? Das überraschende hierbei ist, dass nicht die eigentlich aktiven Einheiten, nämlich die Neuronen, adaptiv sind, sondern die Synapsen, das heißt die Verbindungen. Diese können nämlich ihre Leitfähigkeit verändern. Man weiss, dass eine Synapse umso mehr gestärkt wird, je mehr Spannungsimpulse sie übertragen muss. Stärker heißt hier, dass die Synapse eine höhere Leitfähigkeit besitzt. Synapsen, die viel benutzt werden, erhalten also ein immer größeres Gewicht. Bei Synapsen, die sehr wenig oder gar nicht aktiv sind, nimmt die Leitfähigkeit immer mehr ab. Dies kann bis zum Absterben führen.

Im Gehirn arbeiten alle Neuronen asynchron und parallel, aber mit einer im Vergleich zu einem Computer sehr geringen Geschwindigkeit. Die Zeit für einen Impuls eines Neurons beträgt etwa eine Millisekunde, genauso wie die Zeit für den Ionentransport über den synaptischen Spalt. Die Taktfrequenz der Neuronen liegt also unter einem Kiloherz und ist damit etwa um einen Faktor 10^6 kleiner als auf modernen Computern. Dieser Nachteil wird aber bei vielen komplexen kognitiven Aufgaben, wie zum Beispiel dem Erkennen von Bildern, mehr als ausgeglichen durch den sehr hohen Grad der Parallelverarbeitung im Netzwerk der Nervenzellen.

Die Verbindung zur Außenwelt erfolgt durch sensorische Neuronen, zum Beispiel auf der Retina in den Augen oder auch durch Nervenzellen mit sehr langen Axonen, die vom Gehirn bis zu den Muskeln reichen und so Aktionen wie zum Beispiel die Bewegung eines Beins auslösen können.

Unklar ist aber immer noch, wie durch die erwähnten Prinzipien intelligentes Verhalten möglich wird. Genau wie viele Forscher in der Neuroinformatik werden wir im nächsten Abschnitt versuchen, anhand von Simulationen eines einfachen mathematischen Modells neuronaler Netze zu erklären, wie zum Beispiel Musterkennung möglich wird. Starten wir nun also mit der mathematischen Modellierung.

9.1.1 Das mathematische Modell

Zuerst ersetzen wir die stetige Zeitachse durch eine diskrete Zeitskala. Das Neuron i führt dann in einem Zeitschritt folgende Berechnungen durch. Das „Aufladen“ des Aktivierungspotentials, erfolgt einfach durch Summation der gewichteten Ausgabewerte x_1, \dots, x_n aller eingehenden Verbindungen über die Formel

$$\sum_{j=1}^n w_{ij}x_j.$$

Diese gewichtete Summe wird bei den meisten neuronalen Modellen berechnet. Darauf wird dann eine **Aktivierungsfunktion** f angewendet und das Ergebnis

$$x_i = f \left(\sum_{j=1}^n w_{ij}x_j \right)$$

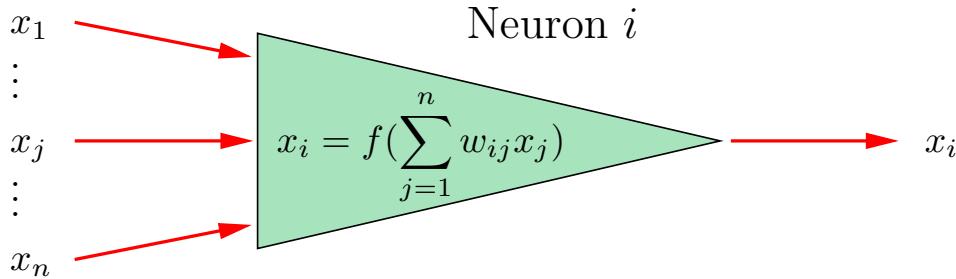
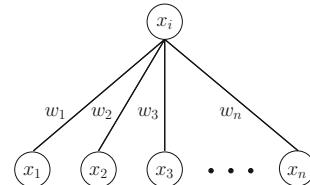


Abb. 9.2 Die Struktur eines formalen Neurons, das auf die gewichtete Summe aller Eingaben die Aktivierungsfunktion f anwendet

Abb. 9.3 Das Neuron mit Stufenfunktion arbeitet wie ein Perzeptron mit Schwelle



als Ausgabe über die synaptischen Gewichte an die Nachbarneuronen weitergegeben. In Abb. 9.2 ist solch ein modelliertes Neuron dargestellt. Für die Aktivierungsfunktion existieren eine Reihe von Möglichkeiten. Die einfachste ist die Identität, das heißt $f(x) = x$. Das Neuron berechnet also nur die gewichtete Summe der Eingabewerte und gibt diese weiter. Dies führt aber häufig zu Konvergenzproblemen bei der neuronalen Dynamik, denn die Funktion $f(x) = x$ ist nicht beschränkt und die Funktionswerte können im Laufe der Zeit über alle Grenzen wachsen.

Sehr wohl beschränkt hingegen ist die Schwellwertfunktion (Heavisidesche Stufenfunktion)

$$H_\Theta(x) = \begin{cases} 0 & \text{falls } x < \Theta \\ 1 & \text{sonst} \end{cases}$$

Das ganze Neuron berechnet dann also seine Ausgabe als

$$x_i = \begin{cases} 0 & \text{falls } \sum_{j=1}^n w_{ij} x_j < \Theta \\ 1 & \text{sonst} \end{cases}$$

Diese Formel ist identisch mit (8.1), das heißt mit einem Perzeptron mit der Schwelle Θ (siehe Abb. 9.3). Die Eingabeneuronen $1, \dots, n$ haben hierbei nur die Funktion von Variablen, die ihre von außen gesetzten Werte x_1, \dots, x_n unverändert weitergeben.

Die Stufenfunktion ist für binäre Neuronen durchaus sinnvoll, denn die Aktivierung eines Neurons kann ohnehin nur die Werte null oder eins annehmen. Für stetige Neuronen

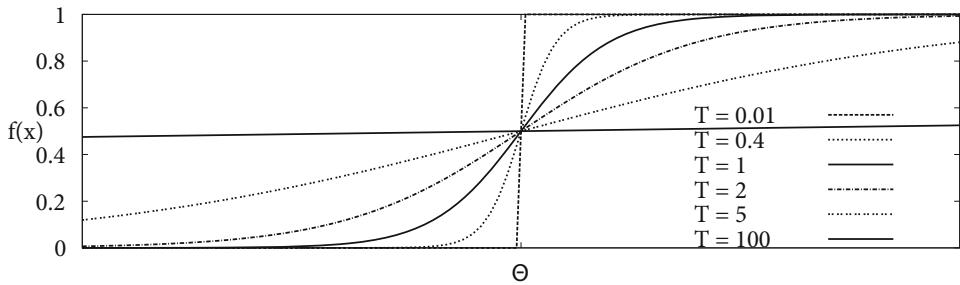


Abb. 9.4 Die Sigmoidfunktion für verschiedene Werte des Parameters T . Man erkennt gut, dass sich im Grenzfall $T \rightarrow 0$ die Stufenfunktion ergibt

mit Aktivierungen zwischen 0 und 1 hingegen erzeugt die Stufenfunktion eine Unstetigkeit. Diese kann man aber glätten durch eine **Sigmoid-Funktion** wie zum Beispiel

$$f(x) = \frac{1}{1 + e^{-\frac{x-\Theta}{T}}}$$

mit dem Graphen in Abb. 9.4. In der Nähe des kritischen Bereichs um die Schwelle Θ verhält sich diese Funktion annähernd linear und sie ist asymptotisch beschränkt. Über den Parameter T lässt sich die Glättung variieren.

Ganz zentral in der Theorie der neuronalen Netze ist die Modellierung des Lernens. Wie schon erwähnt, besteht eine Möglichkeit des Lernens darin, dass eine Synapse umso mehr gestärkt wird, je mehr Spannungsimpulse sie übertragen muss. Dieses Prinzip wurde 1949 von D. Hebb postuliert und bekannt unter dem Namen **Hebb-Regel**:

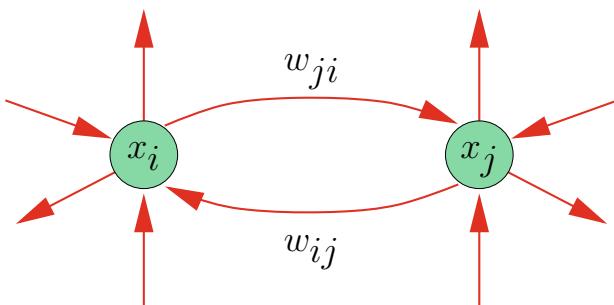
Wenn es eine Verbindung w_{ij} von Neuron j und Neuron i gibt und wiederholt Signale von Neuron j zu Neuron i geschickt werden, was dazu führt, dass die beiden Neuronen gleichzeitig aktiv sind, dann wird das Gewicht w_{ij} verstärkt. Eine mögliche Formel für die Gewichtsänderung Δw_{ij} ist

$$\Delta w_{ij} = \eta x_i x_j$$

mit der Konstante η (Lernrate), welche die Größe der einzelnen Lernschritte bestimmt.

Es gibt viele Modifikationen dieser Regel, die dann zu unterschiedlichen Netzwerktypen oder Lernalgorithmen führen. Im Folgenden werden wir einige davon kennenlernen.

Abb. 9.5 Rekurrente Verbindung von zwei der Neuronen aus einem Hopfield-Netz



9.2 Hopfield-Netze

Betrachtet man die Hebb-Regel, so fällt auf, dass bei Neuronen mit Werten zwischen null und eins die Gewichte im Laufe der Zeit nur wachsen können. Ein Schwächerwerden oder sogar Absterben eines Gewichtes ist nach dieser Regel nicht möglich. Dieses kann man zum Beispiel durch Einführung einer Zerfallskonstante modellieren, die ein unbenutztes Gewicht pro Zeitschritt um einen konstanten Faktor, etwa 0,99, abschwächt.

Ganz anders wird dieses Problem bei dem von Hopfield 1982 vorgestellten Modell gelöst [Hop82]. Es verwendet binäre Neuronen, aber mit den beiden Werten -1 für nicht aktiv und 1 für aktiv. Bei Verwendung der Hebb-Regel erhalten wir also immer, wenn zwei Neuronen gleichzeitig aktiv sind, einen positiven Beitrag zum Gewicht. Wenn aber nur eines der beiden Neuronen aktiv ist, wird Δw_{ij} negativ.

Auf dieser Idee basieren die **Hopfield-Netze**, welche ein sehr schönes und anschauliches Beispiel für einen **Autoassoziativspeicher** darstellen. In einem Autoassoziativspeicher können Muster gespeichert werden. Zum Abrufen der gespeicherten Muster genügt es, ein ähnliches Muster vorzugeben. Der Speicher findet dann das ähnlichste unter den gespeicherten Mustern. Eine klassische Anwendung hierfür stellen Systeme zur Erkennung von Schriftzeichen dar.

In der Lernphase eines Hopfieldnetzes sollen N binär kodierte Muster, gespeichert in den Vektoren q^1, \dots, q^N , gelernt werden. Jede Komponente $q_i^j \in \{-1, 1\}$ solch eines Vektors q^j repräsentiert ein Pixel eines Musters. Bei Vektoren bestehend aus n Pixeln wird nun ein neuronales Netz mit n Neuronen verwendet, eines für jede Pixelposition. Die Neuronen sind vollständig vernetzt mit der Einschränkung, dass die Gewichtsmatrix symmetrisch ist und alle Diagonalelemente $w_{ii} = 0$ sind. Das heißt, es gibt keine Verbindungen von Neuronen mit sich selbst.

Durch die vollständige Vernetzung gibt es komplexe Rückkopplungen, so genannte **Rekurrenzen**, in dem Netzwerk (Abb. 9.5).

Das Lernen der N Muster erfolgt nun ganz einfach durch Berechnung aller Gewichte nach der Formel

$$w_{ij} = \frac{1}{n} \sum_{k=1}^N q_i^k q_j^k. \quad (9.1)$$

Diese Formel weist eine interessante Verwandtschaft zur Hebb-Regel auf. Jedes Muster, bei dem die Pixel i und j den gleichen Wert haben, liefert einen positiven Beitrag zum Gewicht w_{ij} , jedes andere Muster einen negativen. Da jedes Pixel einem Neuron entspricht, werden hier Gewichte zwischen Neuronen gestärkt, die gleichzeitig den gleichen Wert haben. Man beachte den kleinen Unterschied zur Hebb-Regel.

Wenn alle Muster gespeichert sind, kann das Netz für die Mustererkennung verwendet werden. Man legt ein neues Muster x an das Netz an und aktualisiert die Aktivierungen aller Neuronen in einem asynchronen Prozess nach der Regel

$$x_i = \begin{cases} -1 & \text{falls } \sum_{j=1, j \neq i}^n w_{ij} x_j < 0 \\ 1 & \text{sonst} \end{cases} \quad (9.2)$$

so lange, bis das Netz stabil wird, das heißtt, bis sich keine Aktivierungen mehr ändern. Als Programmschema liest sich dies wie folgt

```

HOPFIELDASSOZIATOR( $q$ )
Initialisiere alle Neuronen:  $x = q$ 
Repeat
    i = Random(1,n)
    Aktualisiere Neuron  $i$  nach Gleichung 9.2
Until  $x$  konvergiert
Return( $x$ )

```

9.2.1 Anwendung auf ein Mustererkennungsbeispiel

Wir wenden den beschriebenen Algorithmus auf ein einfaches Mustererkennungsbeispiel an. Es sollen Ziffern in einem 10×10 Pixelfeld erkannt werden. Das Hopfield-Netz hat also 100 Neuronen mit insgesamt

$$\frac{100 \cdot 99}{2} = 4950$$

Gewichten. Zuerst werden die Muster der in Abb. 9.6 oben dargestellten Ziffern 1, 2, 3, 4 trainiert. Das heißtt, die Gewichte werden nach (9.1) berechnet. Dann legen wir verrauschte Muster an und lassen die Hopfield-Dynamik laufen bis zur Konvergenz. In den Zeilen 2 bis 4 der Abbildung sind jeweils fünf Momentaufnahmen der Netzentwicklung beim

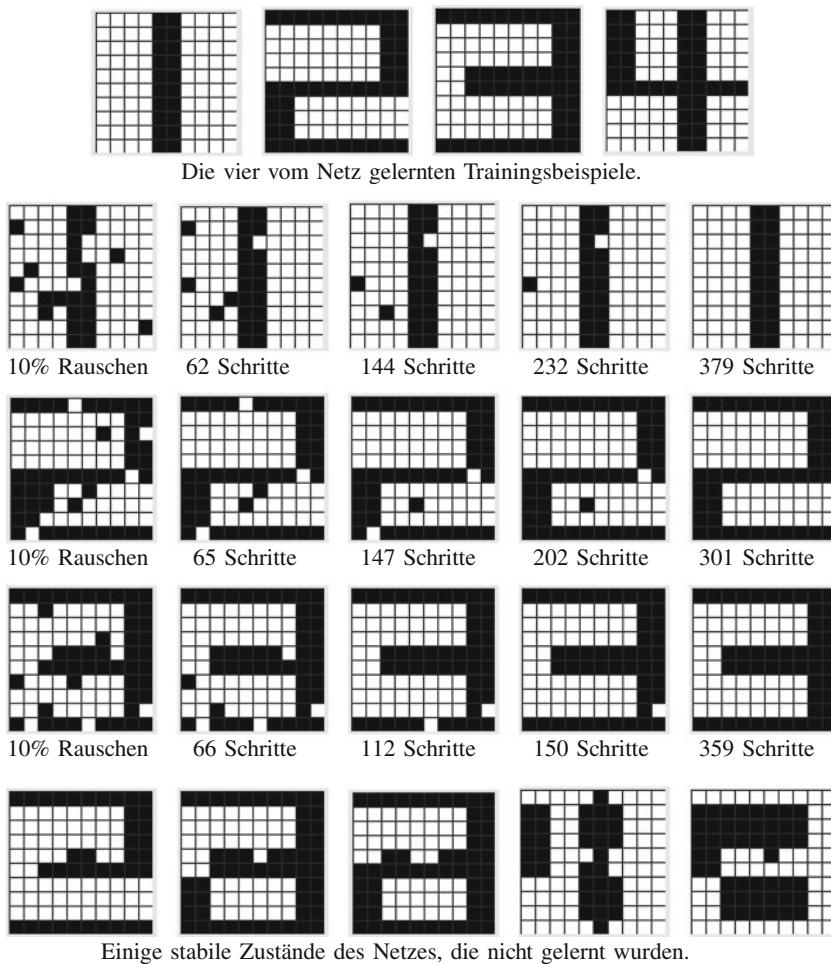


Abb. 9.6 Dynamik eines Hopfield-Netzes. In den Zeilen 2, 3 und 4 erkennt man gut, wie das Netz konvergiert und die gelernten Muster nach etwa 300 bis 400 Schritten erkannt werden. In der letzten Zeile sind einige stabile Zustände dargestellt, die das Netz erreicht, wenn die Eingabemuster zu sehr von allen gelernten Mustern abweichen

Wiedererkennen dargestellt. Bei 10 % Rauschen werden alle vier gelernten Muster sehr zuverlässig erkannt. Ab etwa 20 % Rauschen konvergiert das Verfahren häufig gegen andere gelernte Muster oder sogar gegen Muster, die nicht gelernt wurden. Einige derartige Muster sind in Abb. 9.6 unten gezeigt.

Nun speichern wir in dem gleichen Netz die Ziffern 0 bis 9 (Abb. 9.7 oben) und testen das Netz wieder mit Mustern, die einen zufälligen Anteil von etwa 10 % invertierten Pixeln besitzen. In der Abbildung erkennt man deutlich, dass die Hopfield-Iteration nun auch bei nur 10 % Rauschen oft nicht gegen den ähnlichsten gelernten Zustand konvergiert. Offen-

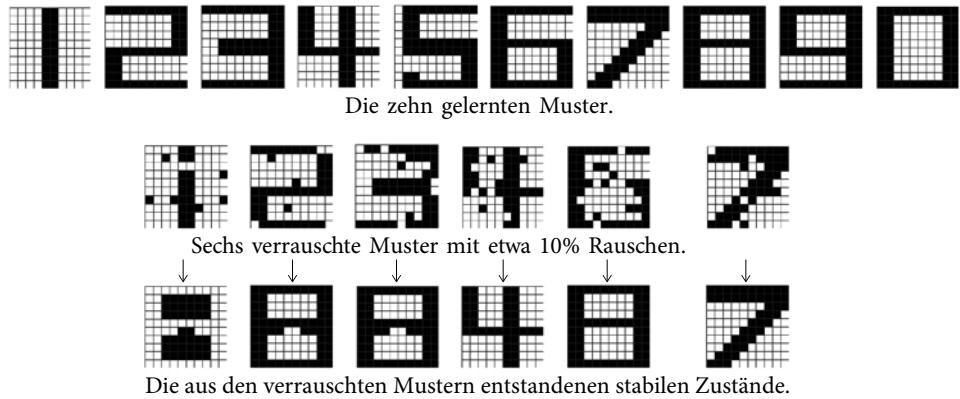


Abb. 9.7 Bei zehn gelernten Zuständen zeigt das Netz chaotisches Verhalten. Schon bei wenig Rauschen konvergiert das Netz gegen falsche Muster oder gegen Artefakte

bar kann das Netz vier Muster sicher speichern und wiedererkennen, bei zehn Mustern hingegen ist die Speicherkapazität überschritten. Um dies besser zu verstehen werden wir einen kurzen Blick auf die Theorie dieser Netze werfen.

9.2.2 Analyse

John Hopfield zeigte 1982 in [Hop82], dass dieses Modell formal äquivalent ist zu einem physikalischen Modell des Magnetismus. Kleine Elementarmagnete, so genannte Spins, beeinflussen sich gegenseitig über ihr Magnetfeld (siehe Abb. 9.8). Wenn man nun zwei solche Spins i und j betrachtet, so wechselwirken diese magnetisch über eine Konstante w_{ij} und die Gesamtenergie des Systems ist dann

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} x_i x_j.$$

Auch in der Physik gilt übrigens $w_{ii} = 0$, denn die Teilchen haben keine Selbstwechselwirkung. Weil physikalische Wechselwirkungen symmetrisch sind, gilt auch $w_{ij} = w_{ji}$.

Ein physikalisches System im Gleichgewicht nimmt einen (stabilen) Zustand minimaler Energie ein und minimiert somit $E(\mathbf{x}, \mathbf{y})$. Wird solch ein System in einen beliebigen Zustand gebracht, so bewegt es sich auf einen Zustand minimaler Energie zu. Die in (9.2) definierte Hopfield-Dynamik entspricht genau diesem Prinzip, denn sie aktualisiert in jedem Schritt den Zustand eines Neurons so, dass es von den zwei Zuständen $-1, 1$ den mit kleinerer Gesamtenergie einnimmt. Der Beitrag des Neurons i zur Gesamtenergie ist

$$-\frac{1}{2} x_i \sum_{j \neq i}^n w_{ij} x_j.$$

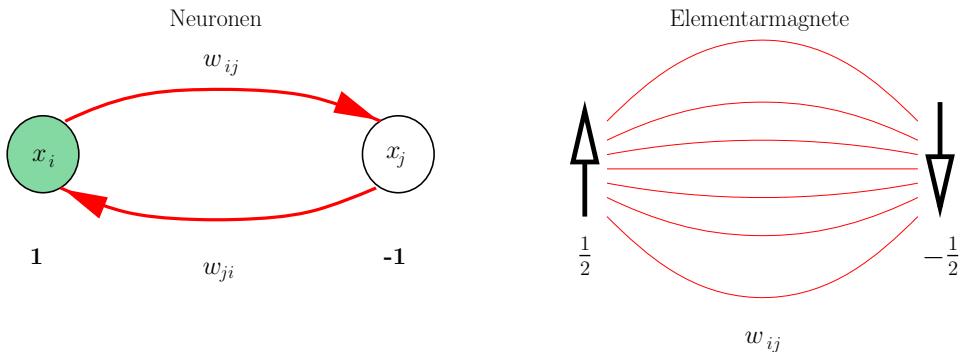


Abb. 9.8 Vergleich zwischen der neuronalen und der physikalischen Interpretation des Hopfield-Modells

Wenn nun

$$\sum_{j \neq i}^n w_{ij} x_j < 0$$

ist, dann führt $x_i = -1$ zu einem negativen Beitrag zur Gesamtenergie, $x_i = 1$ hingegen zu einem positiven Beitrag. Mit $x_i = -1$ nimmt das Netz in diesem Fall einen Zustand niedrigerer Energie an als mit $x_i = 1$. Analog kann man argumentieren, dass im Fall

$$\sum_{j \neq i}^n w_{ij} x_j \geq 0$$

$x_i = 1$ sein muss.

Wenn also jeder einzelne Schritt der neuronalen Dynamik zu einer Verringerung der Energiefunktion führt, so nimmt die gesamte Energie des Systems im Laufe der Zeit monoton ab. Da es nur endlich viele Zustände gibt, bewegt sich das Netzwerk im Laufe der Zeit auf einen Zustand minimaler Energie zu. Die spannende Frage ist nun: Welche Bedeutung haben diese Minima der Energiefunktion?

Wie wir im Mustererkennungsexperiment gesehen haben, konvergiert das System im Fall von wenigen gelernten Mustern gegen eines der gelernten Muster. Die gelernten Muster stellen Minima der Energiefunktion im Zustandsraum dar. Falls jedoch zu viele Muster gelernt werden, so konvergiert das System gegen Minima, die keinen gelernten Mustern entsprechen. Es wechselt von einer geordneten Dynamik in eine chaotische.

Genau diesen Prozess haben Hopfield und andere Physiker untersucht und gezeigt, dass es hier tatsächlich einen Phasenübergang gibt, dem eine kritische Anzahl gelernter Muster zugeordnet ist. Überschreitet die Zahl gelernter Muster diesen Wert, so wechselt das System von der geordneten Phase in die chaotische.

In der Physik des Magnetismus gibt es solch einen Übergang vom ferromagnetischen Modus, in dem alle Elementarmagnete bestrebt sind, sich parallel auszurichten, hin zu einem so genannten Spinglas, bei dem die Spins chaotisch wechselwirken. Ein anschaulicheres Beispiel für solch einen physikalischen Phasenübergang ist das Schmelzen eines Eiskristalls. Im Kristall herrscht ein Zustand hoher Ordnung, denn die H₂O-Moleküle sind streng geordnet. Im flüssigen Wasser hingegen ist die Struktur der Moleküle aufgelöst und deren Orte sind eher zufällig.

Beim neuronalen Netz gibt es also einen Phasenübergang vom geordneten Lernen und Wiedererkennen von Mustern hin zum chaotischen Lernen im Fall von zu vielen Mustern, die dann nicht mehr sicher wiederkannt werden können. Man erkennt hier durchaus Parallelen zu Effekten, die wir bei uns selbst gelegentlich beobachten.

Diesen Phasenübergang kann man verstehen [RMS91], wenn man alle Neuronen in einen Musterzustand, zum Beispiel q^1 , bringt und in den zur Aktualisierung von Neuron i relevanten Term $\sum_{j=1, j \neq i}^n w_{ij} q_j$ die gelernten Gewichte aus (9.1) einsetzt, was zu

$$\begin{aligned} \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} q_j^1 &= \frac{1}{n} \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{k=1}^N q_i^k q_j^k q_j^1 = \frac{1}{n} \sum_{\substack{j=1 \\ j \neq i}}^n \left(q_i^1 (q_j^1)^2 + \sum_{k=2}^N q_i^k q_j^k q_j^1 \right) \\ &= q_i^1 + \frac{1}{n} \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{k=2}^N q_i^k q_j^k q_j^1 \end{aligned}$$

führt. Wir sehen hier die i -te Komponente des angelegten Musters plus eine Summe mit $(n-1)(N-1)$ Summanden. Wenn diese Summanden alle statistisch unabhängig sind, kann man die Summe durch eine normalverteilte Zufallsvariable mit Standardabweichung

$$\frac{1}{n} \sqrt{(n-1)(N-1)} \approx \sqrt{\frac{N-1}{n}}$$

beschreiben. Die statistische Unabhängigkeit erreicht man zum Beispiel mit unkorrelierten Zufallsmustern. Die Summe erzeugt dann ein Rauschen, das nicht störend ist, solange $N \ll n$, das heißt, die Zahl der gelernten Muster viel kleiner als die Zahl der Neuronen bleibt. Wird jedoch $N \approx n$, so ist der Einfluss des Rauschens gleich groß wie das Muster und das Netz reagiert chaotisch. Eine genauere Berechnung des Phasenübergangs ergibt als kritischen Punkt $N = 0,146n$. Angewendet auf unser Beispiel hieße dies, dass bei 100 Neuronen bis zu 14 Muster gespeichert werden können. Da die Muster im Beispiel aber stark korreliert sind, liegt der kritische Wert deutlich tiefer, offenbar zwischen 0,04 und 0,1. Auch ein Wert von 0,146 liegt noch weit unter der Speicherkapazität eines klassischen Listenspeichers (Aufgabe 9.3).

Hopfield-Netze in der vorgestellten Form arbeiten nur dann gut, wenn Muster mit etwa 50 % 1-Bits gelernt werden. Sind die Bits sehr asymmetrisch verteilt, so müssen die Neuronen mit einer Schwelle versehen werden. [Roj93] Im Physikalischen Analogon entspricht

dies dem Anlegen eines äußeren Magnetfeldes, welches auch 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 zur Musterkennung trug das Hopfield Modell in den achtziger Jahren wesentlich zur Welle der Begeisterung für neuronale Netze und zum Aufschwung der Neuroinformatik als wichtiges Teilgebiet der KI bei.² In der Folge wurden viele weitere Netzwerkmodelle entwickelt. Einerseits wurden Netze ohne Rückkopplungen untersucht, denn deren Dynamik ist wesentlich einfacher zu verstehen als die rekurrenten Hopfield-Netze. Andererseits wurde versucht, die Speicherkapazität der Netze zu verbessern, worauf wir im nächsten Abschnitt eingehen werden.

Ein spezielles Problem vieler neuronaler Modelle wurde schon am Hopfield-Modell deutlich. Auch wenn es eine Konvergenzgarantie gibt, so ist nicht sicher, ob das Netz in einen gelernten Zustand konvergiert oder in einem lokalen Minimum hängen bleibt. Ein Versuch, dieses Problem zu lösen, war die Entwicklung der Boltzmann-Maschine mit stetigen Aktivierungswerten und einer probabilistischen Aktualisierungsregel für die Netzwerkdynamik. Mittels eines „Temperatur“-Parameters kann man hier den Anteil zufälliger Zustandsänderungen variieren und so versuchen, lokalen Minima zu entkommen, mit dem Ziel, ein stabiles globales Minimum zu finden. Dieses Verfahren wird als „simulated annealing“ bezeichnet. Annealing heißt Anlassen und meint den Prozess der Wärmebehandlung von Metallen nach dem Härteten mit dem Ziel, das Metall zäher und „stabiler“ zu machen.

Das Hopfield-Modell führt eine Suche nach einem Minimum der Energiefunktion im Raum der Aktivierungswerte durch. Es findet dadurch die in den Gewichten gespeicherten – und damit in der Energiefunktion repräsentierten – Muster wieder. Die Hopfield-Dynamik lässt sich auch auf andere Energiefunktionen anwenden, solange die Gewichtsmatrix symmetrisch ist und die Diagonalelemente null sind. Dies wurde von Hopfield und Tank am Beispiel des Problems des Handlungsreisenden erfolgreich demonstriert [HT85, Zel94]. Aufgabe ist es hier, bei n gegebenen Städten mit Entfernungsmatrix eine kürzeste Rundreise zu finden, die jede Stadt genau einmal besucht.

9.3 Neuronale Assoziativspeicher

Ein klassischer Listenspeicher kann im einfachsten Fall eine Textdatei sein, in der Zeichenketten zeilenweise gespeichert sind. Ist die Datei zeilenweise sortiert, so ist die Suche nach einem Element in logarithmischer Zeit, das heißt, auch bei sehr großen Dateien sehr schnell möglich.

² Auch der Autor wurde von dieser Welle erfasst, die ihn 1987 von der Physik in die KI führte.

Mit Listenspeichern lassen sich aber auch Abbildungen realisieren. Ein Telefonbuch zum Beispiel stellt eine Abbildung von der Menge aller eingetragenen Namen auf die Menge aller Telefonnummern dar. Realisiert ist diese Abbildung durch eine simple Tabelle, typischerweise gespeichert in einer Datenbank.

Eine ähnliche Aufgabe stellt sich zum Beispiel bei der Zugangskontrolle zu einem Gebäude mittels Foto des Gesichts der zu kontrollierenden Person. Man könnte hier auch mit einer Datenbank arbeiten, in der von jeder Person ein Foto zusammen mit dem Namen und eventuell weiteren Daten gespeichert ist. Die Kamera am Eingang nimmt dann ein Foto der Person auf und sucht in der Datenbank nach einem identischen Foto. Falls das Foto gefunden wird, ist die Person identifiziert und erhält Zugang zu dem Gebäude. Allerdings wird ein Gebäude mit solch einem Kontrollsysteem nicht viele Besuche erhalten, denn die Wahrscheinlichkeit, dass das aktuelle Foto exakt mit dem gespeicherten übereinstimmt, ist sehr klein.

Hier genügt das bloße Speichern der Fotos in einer Tabelle nicht. Vielmehr wird ein **Assoziativspeicher** benötigt, der in der Lage ist, nicht nur das gespeicherte Foto dem richtigen Namen zuzuordnen, sondern jedes aus einer eventuell unendlichen Menge von „ähnlichen“ Fotos dem richtigen Namen zuordnen kann. Eine typische Aufgabe also für maschinelle Lernverfahren. Aus einer endlichen Menge von Trainingsdaten, nämlich den gespeicherten Fotos zusammen mit den Namen, soll solch eine Ähnlichkeitserhaltende Funktion generiert werden. Ein einfacher Ansatz hierfür wäre die in Abschn. 8.3 vorgestellte Nearest Neighbour Methode. Beim Lernen werden einfach nur alle Fotos gespeichert.

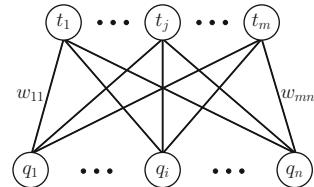
Zur Anwendung der Funktion muss dann zu dem aktuellen Foto in der Datenbank das ähnlichste gefunden werden. Bei einer Datenbank mit vielen hochauflösenden Fotos wird dieser Prozess, je nach verwendetem Abstandsmaß, sehr viel Rechenzeit in Anspruch nehmen und ist daher in dieser einfachen Form nicht realisierbar. Man wird daher hier statt solch eines faulen Lernverfahrens eines bevorzugen, bei dem im Lernprozess die Daten in eine Funktion überführt werden, welche dann bei der Anwendung eine sehr schnelle Assoziation herstellt.

Ein weiteres Problem stellt das Finden eines geeigneten Abstandsmaßes dar. Bei Fotos möchte man zum Beispiel, dass eine Person auch erkannt wird, wenn das Gesicht an einer anderen Stelle auf dem Foto erscheint (Translation), wenn es kleiner oder größer oder sogar gedreht ist. Auch kann der Blickwinkel ein anderer sein oder die Beleuchtung variieren.

Hier zeigen die neuronalen Netze ihre Stärken. Ohne dass der Entwickler sich Gedanken machen muss über ein passendes Ähnlichkeitsmaß, liefern sie schon gute Resultate. Wir werden zwei der einfachsten Assoziativspeichermodelle vorstellen und starten mit einem Modell von Teuvo Kohonen, einem der Pioniere auf diesem Gebiet.

Das im vorigen Abschnitt vorgestellte Hopfield-Modell wäre hier aus zwei Gründen nur schwer anwendbar. Erstens stellt es nur einen **Autoassoziativspeicher**, das heißt eine näherungsweise identische Abbildung, dar, die ähnliche Objekte auf das gelernte Original abbildet. Zweitens ist die komplexe rückgekoppelte Dynamik in der Praxis oft nur schwer handhabbar. Daher betrachten wir nun einfache zweilagige vorwärtsgerichtete (engl. feed-forward) Netze.

Abb. 9.9 Darstellung des Kohonen-Assoziativspeichers als zweilagiges neuronales Netz



9.3.1 Korrelationsmatrix-Speicher

In [Koh72] stellt Kohonen ein Assoziativspeichermodell vor, das auf elementarer linearer Algebra basiert. Dieses bildet Anfragevektoren $\mathbf{x} \in \mathbb{R}^n$ auf Antwortvektoren $\mathbf{y} \in \mathbb{R}^m$ ab. Gesucht ist nun eine Matrix \mathbf{W} , die aus einer Menge

$$T = \{(\mathbf{q}^1, \mathbf{t}^1), \dots, (\mathbf{q}^N, \mathbf{t}^N)\}$$

von Trainingsdaten mit N Anfrage-Antwort-Paaren alle Anfragevektoren korrekt auf ihre Antworten abbildet.³ Das heißt, für $p = 1, \dots, N$ muss

$$\mathbf{t}^p = \mathbf{W} \cdot \mathbf{q}^p, \quad (9.3)$$

beziehungsweise

$$t_i^p = \sum_{j=1}^n w_{ij} q_j^p \quad (9.4)$$

gelten. Zur Berechnung der Matrixelemente w_{ij} wird die Vorschrift

$$w_{ij} = \sum_{p=1}^N q_j^p t_i^p \quad (9.5)$$

verwendet. Diese beiden linearen Gleichungen lassen sich einfach als neuronales Netz verstehen, wenn wir, wie in Abb. 9.9 dargestellt, ein zweilagiges Netz mit \mathbf{q} als Eingabeschicht und \mathbf{t} als Ausgabeschicht definieren. Die Neuronen der Ausgabeschicht besitzen eine lineare Aktivierungsfunktion und als Lernregel wird (9.5) verwendet, was genau der Hebb-Regel entspricht.

Bevor wir zeigen, dass das Netz die Trainingsdaten wiedererkennt, benötigen wir noch folgende Definition:

³ Für eine deutliche Unterscheidung zwischen Trainingsdaten und anderen Werten eines Neurons werden wir im Folgenden die Anfragevektoren immer mit \mathbf{q} (von engl. query) und die gewünschten Antwortvektoren mit \mathbf{t} (von engl. target) bezeichnen.

Definition 9.1

Zwei Vektoren \mathbf{x} und \mathbf{y} heißen **orthonormal**, wenn

$$\mathbf{x} \cdot \mathbf{y} = \begin{cases} 1 & \text{falls } \mathbf{x} = \mathbf{y} \\ 0 & \text{sonst} \end{cases}$$

Damit gilt

Satz 9.1

Wenn alle N Anfragevektoren \mathbf{q}^p in den Trainingsdaten orthonormal sind, so wird jeder Vektor \mathbf{q}^p durch Multiplikation mit der Matrix w_{ij} aus (9.5) auf den entsprechenden Antwortvektor \mathbf{t}^p abgebildet.

Beweis Wir setzen (9.5) in (9.4) ein und erhalten

$$\begin{aligned} (\mathbf{W} \cdot \mathbf{q}^p)_i &= \sum_{j=1}^n w_{ij} q_j^p = \sum_{j=1}^n \sum_{r=1}^N q_j^r t_i^r q_j^p = \sum_{j=1}^n \left(q_j^p q_j^p t_i^p + \sum_{\substack{r=1 \\ r \neq p}}^N q_j^r q_j^p t_i^r \right) \\ &= t_i^p \underbrace{\sum_{j=1}^n q_j^p q_j^p}_{=1} + \underbrace{\sum_{\substack{r=1 \\ r \neq p}}^N t_i^r \sum_{j=1}^n q_j^r q_j^p}_{=0} = t_i^p. \quad \square \end{aligned}$$

Da lineare Abbildungen stetig und injektiv sind, wissen wir, dass die Abbildung von Anfragevektoren auf Antwortvektoren Ähnlichkeitserhaltend ist. Ähnliche Anfragen werden also wegen der Stetigkeit auf ähnliche Antworten abgebildet. Gleichzeitig wissen wir aber auch, dass unterschiedliche Anfragen auf unterschiedliche Antworten abgebildet werden. Wurde also zum Beispiel das Netz trainiert, Gesichter auf Namen abzubilden und sei der Name Hans einem Gesicht zugeordnet, so ist sicher, dass bei Eingabe eines ähnlichen Gesichtes eine zu „Hans“ ähnliche Ausgabe, aber garantiert nicht „Hans“, berechnet wird. Wenn die Ausgabe überhaupt als Zeichenkette interpretierbar ist, so allenfalls zum Beispiel als „Gans“ oder „Hbns“. Um eine Assoziation zum ähnlichsten gelernten Fall zu erreichen, benutzt Kohonen eine binäre Kodierung der Ausgabeneuronen beim Lernen. Wenn dann als Ergebnis einer Anfrage Werte berechnet werden, die nicht null oder eins sind, so werden diese gerundet. Auch damit hat man aber noch keine Garantie, einen der gelernten Antwortvektoren zu treffen. Alternativ wäre auch eine nachgeschaltete Abbildung der berechneten Antwort auf den gelernten Antwortvektor mit dem kleinsten Abstand denkbar.

9.3.2 Die Pseudoinverse

Zur Berechnung der Gewichtsmatrix \mathbf{W} kann man aber auch anders vorgehen, indem man alle Anfragevektoren als Spalten einer $n \times N$ Matrix $\mathbf{Q} = (\mathbf{q}^1, \dots, \mathbf{q}^N)$ und analog die Antwortvektoren als Spalten einer $m \times N$ Matrix $\mathbf{T} = (\mathbf{t}^1, \dots, \mathbf{t}^N)$ auffasst. Damit lässt sich (9.3) in die Form

$$\mathbf{T} = \mathbf{W} \cdot \mathbf{Q} \quad (9.6)$$

bringen. Nun versucht man diese Gleichung nach \mathbf{W} aufzulösen. Formal stellt man um und erhält

$$\mathbf{W} = \mathbf{T} \cdot \mathbf{Q}^{-1}. \quad (9.7)$$

Voraussetzung für diese Umstellung ist die Invertierbarkeit von \mathbf{Q} . Dazu muss die Matrix quadratisch sein und aus linear unabhängigen Spaltenvektoren bestehen. Das heißt, es muss $n = N$ gelten und die n Anfragevektoren müssen alle linear unabhängig sein. Diese Bedingung ist zwar unangenehm, aber immer noch nicht ganz so streng wie die oben geforderte Orthonormalität. Das Lernen nach der Hebb-Regel hat aber den Vorteil, dass man diese auch dann einfach anwenden kann, wenn die Anfragevektoren nicht orthonormal sind, in der Hoffnung, dass der Assoziator trotzdem passabel arbeitet. Dies ist hier nicht so einfach, denn wie soll man eine nicht invertierbare Matrix invertieren?

Eine Matrix \mathbf{Q} ist invertierbar genau dann, wenn es eine Matrix \mathbf{Q}^{-1} gibt mit der Eigenschaft

$$\mathbf{Q} \cdot \mathbf{Q}^{-1} = \mathbf{I}, \quad (9.8)$$

wobei \mathbf{I} die Einheitsmatrix ist. Wenn nun \mathbf{Q} nicht invertierbar ist, zum Beispiel weil \mathbf{Q} nicht quadratisch ist, so gibt es keine Matrix \mathbf{Q}^{-1} mit dieser Eigenschaft. Es gibt aber sicher eine Matrix, die dieser Eigenschaft am nächsten kommt. In diesem Sinne definieren wir

Definition 9.2

Sei \mathbf{Q} eine reelle $n \times m$ Matrix. Eine $m \times n$ Matrix \mathbf{Q}^+ heißt **Pseudoinverse** zu \mathbf{Q} , wenn sie

$$\|\mathbf{Q} \cdot \mathbf{Q}^+ - \mathbf{I}\|$$

minimiert. Hierbei ist $\|M\|$ die quadratische Norm, das heißt die Summe der Quadrate aller Matrixelemente von M .

Nun können wir mittels

$$\mathbf{W} = \mathbf{T} \cdot \mathbf{Q}^+ \quad (9.9)$$

eine Gewichtsmatrix berechnen, die den Assoziationsfehler (engl. crosstalk) $\mathbf{T} - \mathbf{W} \cdot \mathbf{Q}$ minimiert. Das Finden der Pseudoinversen ist nicht einfach. Ein Verfahren hierzu werden wir in Form des Backpropagation-Algorithmus in Abschn. 9.5 kennenlernen.

9.3.3 Die binäre Hebb-Regel

Im Kontext der Assoziativspeicher wurde auch eine so genannte binäre Hebb-Regel vorgeschlagen. Dabei wird vorausgesetzt, dass die Muster binär kodiert sind. Das heißt, für alle Muster gilt $\mathbf{q}^p \in \{0, 1\}^n$ und $\mathbf{t}^p \in \{0, 1\}^m$. Außerdem wird die Summation aus (9.5) ersetzt durch ein schlichtes logisches Oder und man erhält die binäre Hebb-Regel

$$w_{ij} = \bigvee_{p=1}^N q_j^p t_i^p. \quad (9.10)$$

Die Gewichtsmatrix ist damit auch binär, und ein Matrixelement w_{ij} ist genau dann gleich Eins, wenn mindestens einer der Beiträge $q_j^1 t_i^1, \dots, q_j^N t_i^N$ nicht null ist. Alle anderen Matrixelemente sind null. Man ist versucht zu glauben, dass hier beim Lernen viel Information verloren geht, denn wenn ein Matrixelement einmal den Wert 1 angenommen hat, bewirken weitere Muster keine Änderung mehr.

An einem Beispiel mit $n = 10, m = 6$ ist in Abb. 9.10 gezeigt, wie nach dem Lernen von drei Paaren $(\mathbf{q}^p, \mathbf{t}^p)$ die Matrix mit Einsen besetzt ist.

Zum Abrufen der gespeicherten Muster multiplizieren wir einfach einen Anfragevektor \mathbf{q} mit der Matrix und betrachten das Ergebnis $\mathbf{W}\mathbf{q}$. Wir testen dies am Beispiel und erhalten Abb. 9.11.

Man erkennt, dass in den Antwortvektoren auf der rechten Seite an den Stellen der Wert 3 steht, an denen der gelernte Antwortvektor eine Eins hatte. Durch eine Schwellwertbildung mit der Schwelle 3 würden wir die richtigen Resultate erhalten. Als Schwelle wählen wir im allgemeinen Fall die Zahl der Einsen im Anfragevektor. Jedes Ausgabeneuron arbeitet also wie ein Perzeptron, allerdings mit variabler Schwelle.

Solange die Gewichtsmatrix dünn besetzt ist, geht dieses Verfahren gut. Wenn viele unterschiedliche Muster gespeichert werden, wird die Matrix aber immer voller. Im Extremfall enthält sie nur noch Einsen. Dann bestehen alle Antworten nach der Schwellwertbildung aus lauter Einsen und enthalten keine Information mehr.

Dieser Fall tritt selten auf solange die Zahl der Bits, die in der Matrix gespeichert sind, nicht zu groß wird. Die Matrix hat eine Größe von $m \cdot n$ Elementen. Ein zu speicherndes Paar hat $m + n$ Bits. Man kann zeigen [Pal80], dass die Zahl der speicherbaren Muster

Abb. 9.10 Die Matrix W nach dem Speichern von drei Paaren $(q^1, t^1), (q^2, t^2), (q^3, t^3)$. Leere Felder entsprechen dem Wert 0

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

Abb. 9.11 Berechnung der Produkte Wq^1, Wq^2, Wq^3

q^3			1	1	1			
q^2	1	1					1	
q^1		1			1	1		
	1	1				1		2
							0	0
	1			1	1		3	2
	1	1	1	1	1	1	3	3
		1	1	1			1	0
							0	0

N_{\max} durch folgende Bedingung bestimmt wird:

$$\alpha = \frac{\text{Zahl speicherbarer Bits}}{\text{Zahl der Speicherzellen}} = \frac{(m+n)N_{\max}}{m n} \leq \ln 2 \approx 0,69. \quad (9.11)$$

Für einen Listenspeicher ist $\alpha = 1$. Der Assoziativspeicher mit binärer Hebb-Regel hat eine maximale Spechtereffizienz von $\alpha = 0,69$ im Vergleich zu $\alpha = 0,72$ beim Kohonen-Assoziativspeicher und $\alpha = 0,292$ beim Hopfield-Netz [Pal80, Pal91]. Die Speicherkapazität mit binärer Hebb-Regel ist also im Vergleich zu dem Kohonen-Modell mit stetigen Neuronen erstaunlich hoch.

Es ist offensichtlich, dass solch ein Speicher weniger schnell „voll“ wird, wenn die Anfrage- und Antwortvektoren spärlich mit Einsen besetzt sind. Nicht nur aus diesem Grund ist die Kodierung von Ein- und Ausgabe bei Assoziativspeichern – wie auch bei anderen neuronalen Netzen – für eine gute Leistungsfähigkeit sehr wichtig. Dies demonstrieren wir nun an einer Anwendung dieses Speichers mit geeignet gewählter Kodierung der Ein- und Ausgaben.

9.3.4 Ein Fehlerkorrekturprogramm

Als Anwendung für den beschriebenen Assoziativspeicher mit binärer Hebb-Regel wählen wir ein Programm, das fehlerhafte Eingaben korrigiert und auf gespeicherte Worte aus einem Lexikon abbildet [Ben88]. Offenbar würde hier ein Autoassoziativspeicher benötigt. Da wir aber die Anfrage- und Antwortvektoren unterschiedlich kodieren, trifft dies nicht zu. Für die Anfragevektoren q wählen wir eine Paarkodierung. Bei einem Alphabet mit 26 Zeichen gibt es $26 \cdot 26 = 676$ geordnete Paare von Buchstaben. Der Anfragevektor besitzt mit 676 Bits für jedes der möglichen Paare

$$\text{aa, ab, ..., az, ba, ..., bz, ..., za, ..., zz}$$

ein Bit. Kommt ein Paar aus Buchstaben in dem Wort vor, so wird an der entsprechenden Stelle eine Eins eingetragen. Für das Wort „hans“ etwa werden die Stellen für „ha“, „an“ und „ns“ mit Eins besetzt. Beim Antwortvektor t werden für Worte mit einer maximalen Länge (zum Beispiel 10 Zeichen) für jede Position in dem Wort 26 Bit reserviert. Für den i -ten Buchstaben im Alphabet an Position j im Wort wird dann das Bit Nummer $(j - 1) \cdot 26 + i$ gesetzt. Für das Wort „hans“ werden dann die Bits Nummer 8, 27, 66 und 97 besetzt. Bei maximal 10 Zeichen pro Wort hat der Antwortvektor also eine Länge von 260 Bit.

Die Gewichtsmatrix W hat damit eine Größe von $676 \cdot 260$ Bit = 199.420 Bit. Darin können nach (9.11) höchstens etwa

$$N_{\max} \leq 0,69 \frac{mn}{m+n} = 0,69 \frac{676 \cdot 260}{676 + 260} \approx 130$$

Worte gespeichert werden. Wir speichern mit 72 Vornamen gut halb so viele und testen das System. In Abb. 9.12 sind die gespeicherten Namen angegeben und für einige Beispieleingaben die Ausgaben des Programms. Die Schwelle wird zu Beginn immer auf die Zahl der Bits in der kodierten Anfrage gesetzt. Dies ist hier die Zahl der Buchstabenpaare, also Wortlänge minus eins. Dann wird sie reduziert bis auf 2. Man könnte die Wahl der Schwelle noch weiter automatisieren, indem bei jeder versuchten Schwelle mit dem Wörterbuch verglichen und bei Erfolg das gefundene Wort ausgegeben wird.

Interessant ist die Reaktion auf die mehrdeutige Eingabe „andr“ sowie auf „johanne“. In beiden Fällen bildet das Netz eine Mischung aus zwei gespeicherten passenden Wörtern. Man erkennt eine wichtige Stärke der neuronalen Netze. Sie sind in der Lage, ohne ein explizites Ähnlichkeitsmaß Assoziationen zu ähnlichen Objekten herzustellen. Allerdings gibt es keine Garantie für die „richtige“ Lösung, ähnlich wie bei der heuristischen Suche oder bei menschlichen Entscheidungen.

Da bei allen bisher vorgestellten neuronalen Modellen die Trainingsdaten in Form von Eingabe-Ausgabe-Paaren vorliegen müssen, handelt es sich hier um Lernen mit Lehrer, genauso wie bei den in den folgenden Abschnitten vorgestellten Netzen.

Gespeicherte Namen:

agathe, agnes, alexander, andreas, andree, anna, annemarie, astrid, august, bernhard, bjorn, cathrin, christian, christoph, corinna, corrado, dieter, elisabeth, elvira, erdmut, 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

Assoziationen des Programms:

Gib Muster: harry	Gib Muster: andrees
Schwelle: 4, Antwort: harry	Schwelle: 6, Antwort: a
Schwelle: 3, Antwort: harry	Schwelle: 5, Antwort: andree
Schwelle: 2, Antwort: horryrrde	Schwelle: 4, Antwort: andrees
-----	Schwelle: 3, Antwort: mnnrens
Gib Muster: ute	Schwelle: 2, Antwort: morxsnssr
-----	-----
Schwelle: 2, Antwort: ute	Gib Muster: johanne
-----	Schwelle: 6, Antwort: johnnnes
Gib Muster: gerhar	Schwelle: 5, Antwort: johnnnes
Schwelle: 5, Antwort: gerhard	Schwelle: 4, Antwort: jornnnrse
Schwelle: 4, Antwort: gerrarn	Schwelle: 3, Antwort: sorrnryse
Schwelle: 3, Antwort: jerrhrrd	Schwelle: 2, Antwort: wtrrsyrse
Schwelle: 2, Antwort: juryrrde	-----
-----	Gib Muster: johnnnes
Gib Muster: egrhard	Schwelle: 6, Antwort: joh
Schwelle: 6, Antwort:	Schwelle: 5, Antwort: johnnnes
Schwelle: 5, Antwort:	Schwelle: 4, Antwort: johnnyes
Schwelle: 4, Antwort: gerhard	Schwelle: 3, Antwort: jonnyyes
Schwelle: 3, Antwort: gernhrrd	Schwelle: 2, Antwort: jornsyrse
Schwelle: 2, Antwort: irrryrrde	-----
-----	Gib Muster: johnnyes
Gib Muster: andr	Schwelle: 7, Antwort:
Schwelle: 3, Antwort: andrees	Schwelle: 6, Antwort: joh
Schwelle: 2, Antwort: anexenser	Schwelle: 5, Antwort: johnny
	Schwelle: 4, Antwort: johnnyes
	Schwelle: 3, Antwort: johnnyes
	Schwelle: 2, Antwort: johnnyes

Abb. 9.12 Anwendung des Fehlerkorrekturprogramms auf verschiedene gelernte oder fehlerhafte Eingaben. Die korrekten Eingaben werden bei der maximalen, das heißt der ersten versuchten Schwelle gefunden. Bei fehlerhaften Eingaben muss für eine korrekte Assoziation die Schwelle erniedrigt werden

9.4 Lineare Netze mit minimalem Fehler

Die bei den bisher vorgestellten neuronalen Modellen verwendete Hebb-Regel arbeitet mittels Assoziationen zwischen benachbarten Neuronen. Beim Assoziativspeicher wird dies ausgenutzt, um eine Abbildung von Anfragevektoren auf Antworten zu lernen. Dies klappt in vielen Fällen sehr gut, insbesondere wenn zum Beispiel die Anfragevektoren linear unabhängig sind. Ist diese Bedingung jedoch nicht erfüllt, zum Beispiel wenn zu viele Trainingsdaten vorhanden sind, so stellt sich die Frage, wie man nun eine optimale Gewichtsmatrix findet? Optimal heißt, dass sie den mittleren Fehler minimiert.

Wir Menschen sind in der Lage, auch aus Fehlern zu lernen. Diese Möglichkeit bietet die Hebb-Regel nicht. Der im Folgenden beschriebene Backpropagation-Algorithmus verwendet eine elegante, aus der Funktionsapproximation bekannte Lösung, um die Gewichte so zu verändern, dass der Fehler auf den Trainingsdaten minimiert wird.

Seien also N Paare von Trainingsvektoren

$$T = \{(\mathbf{q}^1, \mathbf{t}^1), \dots, (\mathbf{q}^N, \mathbf{t}^N)\}$$

gegeben mit $\mathbf{q}^p \in [0, 1]^n$, $\mathbf{t}^p \in [0, 1]^m$. Gesucht ist eine Funktion $f : [0, 1]^n \rightarrow [0, 1]^m$, welche den mittleren quadratischen Fehler

$$\sum_{p=1}^N (f(\mathbf{q}^p) - \mathbf{t}^p)^2$$

auf den Daten minimiert. Nehmen wir vorerst an, die Daten enthalten keine Widersprüche. Das heißt, es gibt in den Trainingsdaten keinen Anfragevektor, der auf zwei verschiedene Antworten abgebildet werden soll. In diesem Fall ist es nicht schwierig, eine Funktion zu finden, die diese Bedingung erfüllt. Ja, es existieren sogar unendlich viele Funktionen, welche diesen Fehler zu null machen. Wir definieren die Funktion

$$f(\mathbf{q}) = 0, \quad \text{falls } \mathbf{q} \notin \{\mathbf{q}^1, \dots, \mathbf{q}^N\}$$

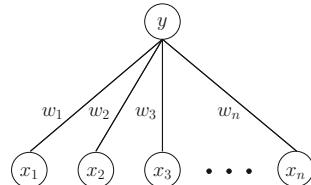
und

$$f(\mathbf{q}^p) = \mathbf{t}^p \quad \forall p \in \{1, \dots, N\}.$$

Dies ist eine Funktion, die den Fehler auf den Trainingsdaten sogar zu null macht. Was wollen wir mehr? Oder: Warum werden wir mit dieser Funktion nicht glücklich?

Die Antwort lautet: Weil wir ein intelligentes System bauen wollen! Intelligent heißt beim Lernen unter anderem, dass die gelernte Funktion gut generalisiert von den Trainingsdaten auf neue unbekannte Daten aus der gleichen repräsentativen Datenmenge. In anderen Worten heißt das: Wir wollen keine Überanpassung (engl. overfitting) an die Daten durch Auswendiglernen. Was wollen wir denn nun wirklich haben?

Abb. 9.13 Ein zweilagiges Netzwerk mit einem Ausgabeneuron



Wir wollen eine Funktion haben, die glatt ist und zwischen den Punkten „ausgleicht“. Sinnvolle Forderungen wären zum Beispiel Stetigkeit und mehrfache stetige Differenzierbarkeit. Da es aber mit diesen Bedingungen immer noch unendlich viele Funktionen gibt, welche den Fehler zu null machen, müssen wir die Funktionenklasse noch weiter einschränken.

9.4.1 Die Methode der kleinsten Quadrate

Die einfachste Wahl ist eine lineare Abbildung. Wir starten mit einem zweilagigen Netzwerk (Abb. 9.13), bei dem das einzige Neuron y der zweiten Lage seine Aktivierung mittels

$$y = f\left(\sum_{i=1}^n w_i x_i\right)$$

mit $f(x) = x$ berechnet. Die Tatsache, dass wir hier nur ein Ausgabeneuron betrachten, stellt keine wirkliche Einschränkung dar, denn ein zweilagiges Netzwerk mit zwei oder mehr Ausgabeneuronen lässt sich immer zerlegen in unabhängige Netze mit je einem der ursprünglichen Ausgabeneuronen und identischen Eingabeneuronen. Die Gewichte der Teilnetze sind alle unabhängig. Die Verwendung einer Sigmoid-Funktion statt der linearen Aktivierung bringt hier keinen Gewinn, denn die Sigmoid-Funktion ist streng monoton wachsend und ändert nichts an den Größenverhältnissen verschiedener Ausgabewerte untereinander.

Gesucht ist nun also ein Vektor w , der den quadratischen Fehler

$$E(w) = \sum_{p=1}^N (\mathbf{w} \mathbf{q}^p - t^p)^2 = \sum_{p=1}^N \left(\sum_{i=1}^n w_i q_i^p - t^p \right)^2$$

minimiert. Eine notwendige Bedingung für ein Minimum dieser Fehlerfunktion ist das Verschwinden aller partiellen Ableitungen. Also fordern wir, dass für $j = 1, \dots, n$

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^N \left(\sum_{i=1}^n w_i q_i^p - t^p \right) q_j^p = 0$$

wird. Ausmultiplizieren ergibt

$$\sum_{p=1}^N \left(\sum_{i=1}^n w_i q_i^p q_j^p - t^p q_j^p \right) = 0$$

und Vertauschen der Summen führt auf das lineare Gleichungssystem

$$\sum_{i=1}^n w_i \sum_{p=1}^N q_i^p q_j^p = \sum_{p=1}^N t^p q_j^p,$$

das sich mit

$$A_{ij} = \sum_{p=1}^N q_i^p q_j^p \quad \text{und} \quad b_j = \sum_{p=1}^N t^p q_j^p \quad (9.12)$$

als Matrixgleichung

$$A\mathbf{w} = \mathbf{b} \quad (9.13)$$

schreiben lässt. Diese so genannten Normalgleichungen besitzen immer mindestens eine Lösung und wenn A invertierbar ist, genau eine. Außerdem ist die Matrix A positiv definit, was zur Folge hat, dass die gefundene Lösung im eindeutigen Fall ein globales Minimum darstellt. Dieses Verfahren ist bekannt unter dem Namen Methode der kleinsten Quadrate (engl. least mean squares, kurz LMS).

Die Rechenzeit für das Aufstellen der Matrix A wächst wie $\Theta(N \cdot n^2)$ und die Zeit für das Auflösen des Gleichungssystems wie $O(n^3)$. Diese Methode lässt sich in einfacher Weise auf mehrere Ausgabeneuronen erweitern, denn bei zweilagigen vorwärtsgerichteten Netzen sind die Ausgabeneuronen unabhängig voneinander.

9.4.2 Anwendung auf die Appendizitisdaten

Als Anwendung bestimmen wir nun einen linearen Score für die Appendizitisdiagnose. Wir verwenden die aus Abschn. 8.4.5 bekannten Daten des LEXMED-Projektes (Abschn. 7.3), bestimmen daraus mit der Methode der kleinsten Quadrate eine lineare Abbildung von den Symptomen auf die stetige Klassenvariable *AppScore* mit Werten im Intervall $[0, 1]$ und erhalten die Linearkombination

$$\begin{aligned} AppScore = & 0,00085 \text{ Alter} - 0,125 \text{ Geschl} + 0,025 \text{ S1Q} + 0,035 \text{ S2Q} - 0,021 \text{ S3Q} - 0,025 \text{ S4Q} \\ & + 0,12 \text{ AbwLok} + 0,031 \text{ AbwGlo} + 0,13 \text{ Losl} + 0,081 \text{ Ersch} + 0,0034 \text{ RektS} \\ & + 0,0027 \text{ TAxi} + 0,0031 \text{ TRek} + 0,000021 \text{ Leuko} - 0,11 \text{ Diab} - 1,83. \end{aligned}$$

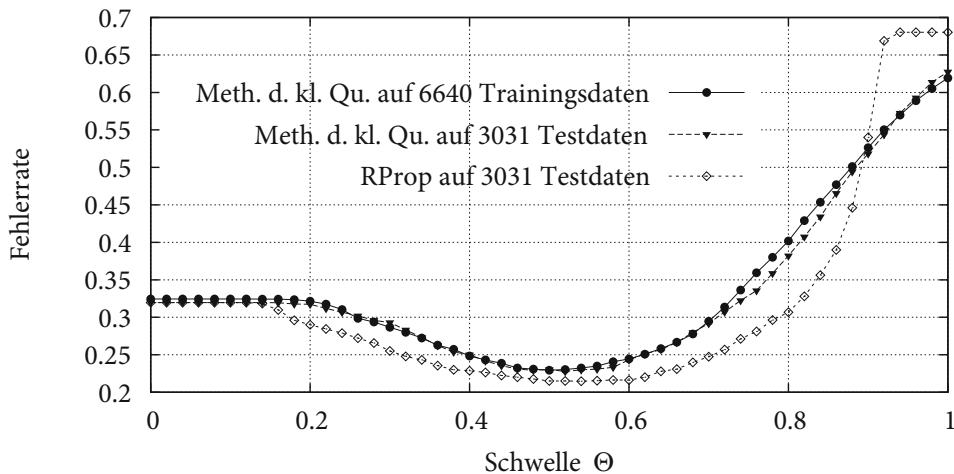


Abb. 9.14 Fehler der Methode der kleinsten Quadrate auf den Trainings- und Testdaten

Diese Funktion liefert stetige Werte für *AppScore*, obwohl die eigentliche binäre Klassenvariable *App* nur die Werte 0 und 1 annimmt. Es muss dann also noch wie beim Perzeptron eine Schwellwertentscheidung erfolgen. Der Klassifikationsfehler des Score in Abhängigkeit von der Schwelle ist in Abb. 9.14 für die Trainingsdaten und die Testdaten aufgetragen. Man erkennt deutlich, dass beide Kurven sich kaum unterscheiden und bei $\Theta = 0,5$ ihr Minimum besitzen. An dem geringen Unterschied der beiden Kurven erkennt man, dass Überanpassung bei dieser Methode kein Problem darstellt, denn das Modell generalisiert sehr gut auf die Testdaten.

In der Abbildung ist außerdem noch das Ergebnis eines nichtlinearen dreilagigen RProp-Netzes (Abschn. 9.5) mit etwas geringerem Fehler bei mittleren Schwellwerten dargestellt. Für die praktische Anwendung des erhaltenen Score und die korrekte Bestimmung der Schwelle Θ ist es wichtig, nicht nur den Fehler zu betrachten, sondern nach der Art der Fehlentscheidungen, nämlich falsch positiv und falsch negativ, zu differenzieren, wie dies bei der LEXMED-Anwendung in Abb. 7.10 erfolgt ist. In der dort dargestellten ROC-Kurve ist auch der hier berechnete Score eingetragen. Man erkennt, dass das einfache lineare Modell dem LEXMED-System deutlich unterlegen ist. Offenbar sind lineare Approximationen für viele komplexe Anwendungen nicht mächtig genug.

9.4.3 Die Delta-Regel

Die Methode der kleinsten Quadrate ist, wie zum Beispiel auch das Perzeptron oder das Lernen von Entscheidungsbäumen, ein so genanntes **Batch-Lernverfahren**, im Unterschied zum **inkrementellen Lernen**. Beim Batch-Lernen müssen alle Trainingsdaten in

einem Lauf gelernt werden. Wenn nun aber neue Trainingsdaten hinzukommen, so können diese nicht einfach „dazugelernt“ werden. Es muss der ganz Lernprozess mit der vergrößerten Menge wiederholt werden. Dieses Problem wird durch inkrementelle Lernverfahren gelöst, die das gelernte Modell an jedes neu hinzukommende Beispiel anpassen können. Bei den im Folgenden betrachteten Verfahren werden wir in jedem Schritt für ein neues Trainingsbeispiel die Gewichte additiv nach der Vorschrift

$$w_j = w_j + \Delta w_j$$

aktualisieren. Um nun eine inkrementelle Variante der Methode der kleinsten Quadrate herzuleiten, betrachten wir nochmal die oben berechneten n partiellen Ableitungen der Fehlerfunktion

$$\frac{\partial E}{\partial w_j} = 2 \sum_{p=1}^N \left(\sum_{i=1}^n w_i q_i^p - t^p \right) q_j^p.$$

Der Gradient

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right)$$

als Vektor aller partiellen Ableitungen der Fehlerfunktion zeigt im n -dimensionalen Raum der Gewichte in die Richtung des stärksten Anstiegs der Fehlerfunktion. Auf der Suche nach einem Minimum folgen wir daher der Richtung des negativen Gradienten und erhalten als Formel für die Änderung der Gewichte

$$\Delta w_j = -\frac{\eta}{2} \frac{\partial E}{\partial w_j} = -\eta \sum_{p=1}^N \left(\sum_{i=1}^n w_i q_i^p - t^p \right) q_j^p,$$

wobei die **Lernrate** η eine frei wählbare positive Konstante ist. Größeres η beschleunigt die Konvergenz, erhöht aber gleichzeitig das Risiko für Oszillationen um Minima oder flache Täler. Daher ist die optimale Wahl von η eine nicht ganz einfache Aufgabe (siehe Abb. 9.15). Oft wird daher mit einem großen η gestartet, etwa $\eta = 1$, welches dann langsam verkleinert wird.

Durch Ersetzen der Aktivierung

$$y^p = \sum_{i=1}^n w_i q_i^p$$

des Ausgabeneurons bei angelegtem Trainingsbeispiel q^p vereinfacht sich die Formel und man erhält die **Delta-Regel**

$$\Delta w_j = \eta \sum_{p=1}^N (t^p - y^p) q_j^p.$$

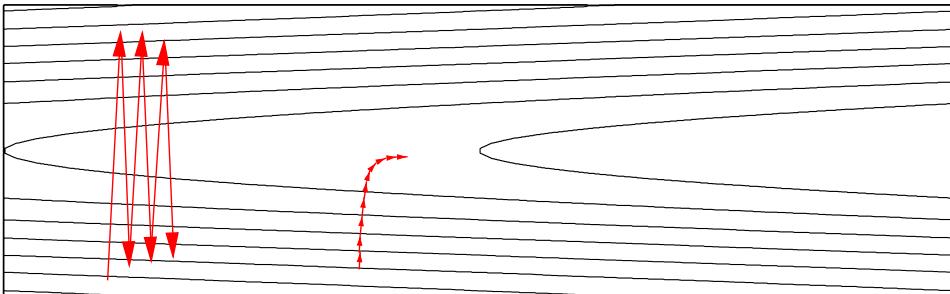


Abb. 9.15 Gradientenabstieg mit großem η (links) und sehr kleinem η (rechts) in einem nach rechts flach abfallenden Tal. Bei zu großem η ergeben sich Oszillationen um das Tal. Bei zu kleinem η hingegen wird die Konvergenz in dem flachen Tal sehr langsam

Abb. 9.16 Das Lernen eines zweilagigen linearen Netzes mit der Deltaregel. Man beachte, dass die Gewichtsänderungen immer erst nach dem Anlegen aller Trainingsbeispiele erfolgen

DELTALERNEN (*Trainingsbeispiele, η*)

Initialisiere alle Gewichte w_j zufällig

Repeat

$$\Delta w = 0$$

For all $(q^p, t^p) \in \text{Trainingsbeispiele}$

Berechne Netzausgabe $y^p = w^p q^p$

$$\Delta w = \Delta w + \eta(t^p - y^p)q^p$$

$$w = w + \Delta w$$

Until w konvergiert

Für jedes Trainingsbeispiel wird also die Differenz aus der gewünschten Antwort t^p und der tatsächlichen Antwort y^p des Netzes mit der angelegten Eingabe q^p berechnet. Nach Aufsummieren über alle Muster werden dann die Gewichte proportional zur Summe geändert. In Abb. 9.16 ist dieser Algorithmus dargestellt.

Man erkennt, dass das Verfahren noch nicht wirklich inkrementell ist, denn die Gewichtsänderungen erfolgen immer erst, nachdem alle Trainingsbeispiele einmal angelegt wurden. Diesen Mangel kann man durch direktes Ändern der Gewichte (inkrementeller Gradientenabstieg) nach jedem Trainingsbeispiel beheben (Abb. 9.17), was jedoch streng genommen keine korrekte Implementierung der Delta-Regel mehr ist.

9.4.4 Vergleich mit dem Perzeptron

Die in Abschn. 8.2.1 vorgestellte Lernregel für das Perzeptron, die Methode der kleinsten Quadrate und auch die Delta-Regel können verwendet werden, um lineare Funktionen aus Daten zu generieren. Beim Perzeptron wird aber, im Gegensatz zu den anderen Methoden, durch die Schwellwertentscheidung ein Klassifizierer für linear separable Klassen gelernt. Die anderen beiden Methoden hingegen erzeugen eine lineare Approximation an die Daten. Wie in Abschn. 9.4.2 gezeigt, kann aus der gelernten linearen Abbildung, falls

Abb. 9.17 Inkrementelle Variante der Delta-Regel

```

DELTALERNENINKREMENTELL (Trainingsbeispiele,  $\eta$ )
Initialisiere alle Gewichte  $w_j$  zufällig
Repeat
    For all  $(q^p, t^p) \in \text{Trainingsbeispiele}$ 
        Berechne Netzausgabe  $y^p = w^p q^p$ 
         $w = w + \eta(t^p - y^p)q^p$ 
    Until  $w$  konvergiert

```

gewünscht, durch Anwendung einer Schwellwertfunktion ein Klassifizierer erzeugt werden.

Das Perzeptron und die Delta-Regel sind iterative Algorithmen, bei denen die Zeit bis zur Konvergenz stark von den Daten abhängt. Im Fall von linear separablen Daten kann die Zahl der Iterationsschritte beim Perzeptron beschränkt werden. Bei der Delta-Regel hingegen gibt es nur eine asymptotische Konvergenzgarantie ohne Schranke [HKP91].

Bei der Methode der kleinsten Quadrate besteht das Lernen im Aufstellen und Auflösen eines linearen Gleichungssystems für den Gewichtsvektor. Es gibt also eine harte Schranke für die Rechenzeit. Deshalb ist die Methode der kleinsten Quadrate immer dann vorzuziehen, wenn das inkrementelle Lernen nicht benötigt wird.

9.5 Der Backpropagation-Algorithmus

Mit dem Backpropagation-Algorithmus stellen wir nun das meist genutzte neuronale Modell vor. Grund für die weite Verbreitung ist die universelle Einsetzbarkeit für beliebige Approximationsaufgaben. Der Algorithmus geht direkt aus der inkrementellen Delta-Regel hervor. Im Unterschied zu dieser wird hier als Aktivierungsfunktion die nichtlineare Sigmoid-Funktion auf die gewichtete Summe der Eingaben angewendet. Außerdem kann ein Backpropagation-Netz über mehr als zwei Schichten von Neuronen verfügen. Bekannt wurde der Algorithmus durch den Artikel [RHR86] in der legendären PDP-Sammlung [RM86].

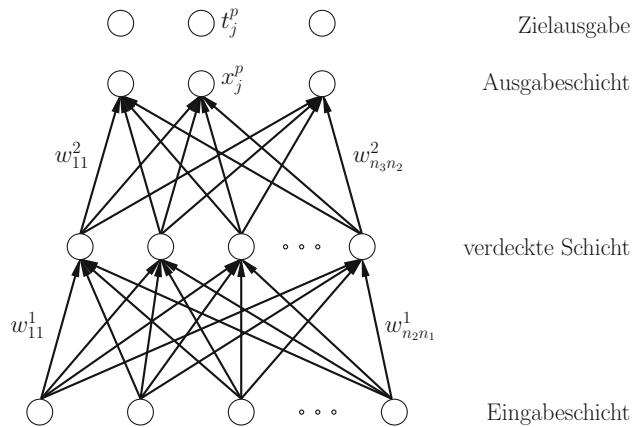
In Abb. 9.18 ist ein typisches Backpropagation-Netz mit einer Eingabeschicht, einer verdeckten Schicht und einer Ausgabeschicht dargestellt. Da die jeweils aktuellen Ausgabewerte x_j^p der Neuronen der Ausgabeschicht mit den Zielausgabewerten t_j^p verglichen werden, sind diese parallel dazu eingezeichnet. Außer den Eingabeneuronen berechnen alle Neuronen ihren aktuellen Wert x_j nach der Vorschrift

$$x_j = f\left(\sum_{i=1}^n w_{ji} x_i\right) \quad (9.14)$$

mit der Sigmoidfunktion

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Abb. 9.18 Ein dreilagiges Backpropagation-Netz mit n_1 Neuronen in der ersten, n_2 Neuronen in der zweiten und n_3 Neuronen in der dritten Schicht



Ähnlich wie bei der inkrementellen Deltaregel werden die Gewichte entsprechend dem negativen Gradienten der über die Ausgabeneuronen summierten quadratischen Fehlerfunktion

$$E_p(\mathbf{w}) = \frac{1}{2} \sum_{k \in \text{Ausgabe}} (t_k^p - x_k^p)^2$$

für das Trainingsmuster p geändert:

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}.$$

Zur Herleitung der Lernregel wird für E_p obiger Ausdruck eingesetzt und darin wird x_k durch (9.14) ersetzt. Darin wiederum kommen rekursiv die Ausgaben x_i der Neuronen der nächst tieferen Schicht vor, und so weiter. Durch mehrfache Anwendung der Kettenregel (siehe [RHR86] oder [Zel94]) erhält man die Backpropagation-Lernregel

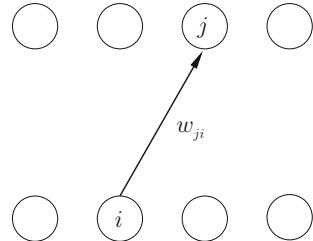
$$\Delta_p w_{ji} = \eta \delta_j^p x_i^p,$$

mit

$$\delta_j^p = \begin{cases} x_j^p (1 - x_j^p) (t_j^p - x_j^p) & \text{falls } j \text{ Ausgabeneuron ist} \\ x_j^p (1 - x_j^p) \sum_k \delta_k^p w_{kj} & \text{falls } j \text{ verdecktes Neuron ist,} \end{cases}$$

welche auch als verallgemeinerte Deltaregel bezeichnet wird. Für alle Neuronen enthält die Formel zur Änderung des Gewichts w_{ji} von Neuron i zu Neuron j (siehe Abb. 9.19) wie

Abb. 9.19 Bezeichnungen der Neuronen und Gewichte für die Anwendung der Backpropagation-Regel



bei der Hebb-Regel einen Term $\eta x_i^p x_j^p$. Der neue Faktor $(1 - x_j^p)$ stellt die bei der Hebb-Regel fehlende Symmetrie zwischen den Aktivierungen 0 und 1 von Neuron j her. Für die Ausgabeneuronen sorgt der Faktor $(t_j^p - x_j^p)$ für eine Gewichtsänderung proportional zum Fehler. Für die verdeckten Neuronen berechnet sich der Wert δ_j^p des Neurons j rekursiv aus allen Änderungen δ_k^p der Neuronen der nächst höheren Schicht.

Der gesamte Ablauf des Lernens ist in Abb. 9.20 dargestellt. Nach dem Berechnen der Netzausgabe (Vorwärtspropagieren) für ein Trainingsbeispiel wird der Approximationfehler berechnet. Dieser wird dann beim Rückwärtspropagieren verwendet, um von Schicht zu Schicht rückwärts die Gewichte zu ändern. Der ganze Prozess wird nun auf alle Trainingsbeispiele angewendet und so lange wiederholt, bis die Gewichte sich nicht mehr verändern oder eine Zeitschranke erreicht ist.

Wenn man ein Netz mit mindestens einer verdeckten Schicht aufbaut, können nicht-lineare Abbildungen gelernt werden. Ohne verdeckte Schicht sind die Ausgabeneuronen trotz der Sigmoidfunktion nicht mächtiger als ein lineares Neuron. Grund hierfür ist die strenge Monotonie der Sigmoidfunktion. Das gleiche gilt für mehrlagige Netze die als Akti-

BACKPROPAGATION (*Trainingsbeispiele, η*)

Initialisiere alle Gewichte w_j zufällig

Repeat

For all $(q^p, t^p) \in$ *Trainingsbeispiele*

1. Anlegen des Anfragevektors q^p an die Eingabeschicht

2. Vorwärtspropagieren:

 Für alle Schichten ab der ersten verdeckten aufwärts

 Für alle Neuronen der Schicht

 Berechne Aktivierung $x_j = f(\sum_{i=1}^n w_{ji} x_i)$

3. Berechnung des quadratischen Fehlers $E_p(w)$

4. Rückwärtspropagieren:

 Für alle Lagen von Gewichten ab der letzten abwärts

 Für jedes Gewicht w_{ji}

$$w_{ji} = w_{ji} + \eta \delta_j^p x_i^p$$

Until w konvergiert oder Zeitschranke erreicht

Abb. 9.20 Der Backpropagation-Algorithmus

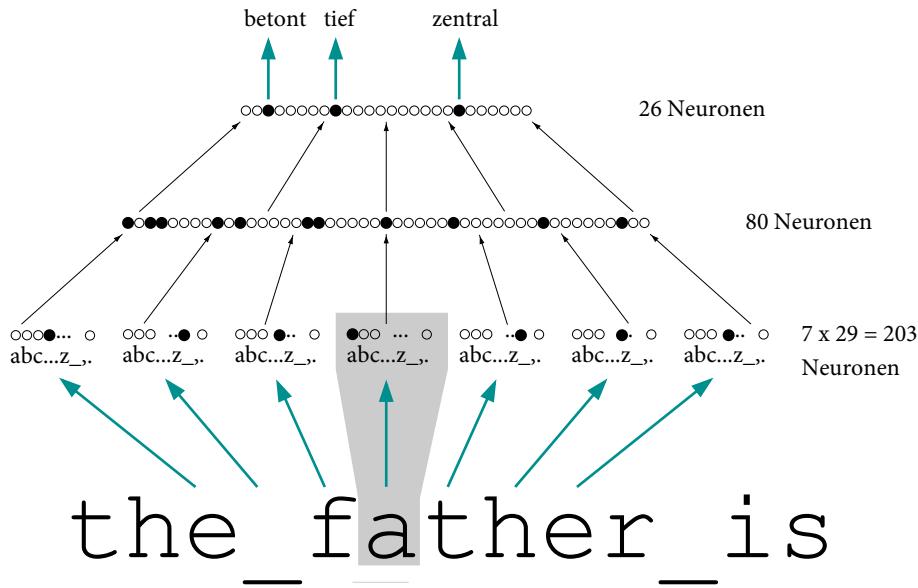


Abb. 9.21 Das NETtalk-Netzwerk bildet einen Text auf seine Ausspracheattribute ab

vierungsfunktion nur eine lineare Funktion, zum Beispiel die Identität, benutzen. Dies liegt in der Linearität des hintereinander Ausführens von linearen Abbildungen begründet.

Genau wie beim Perzepron wird die Klasse der darstellbaren Funktionen auch bei Backpropagation vergrößert, wenn man eine variable Sigmoidfunktion

$$f(x) = \frac{1}{1 + e^{-(x-\Theta)}}.$$

mit Schwellen Θ verwendet. Analog wie in Abschn. 8.2 gezeigt, wird zur Eingabeschicht und zu jeder verdeckten Schicht ein Neuron hinzugefügt, dessen Aktivierung immer den Wert eins hat und das mit allen Neuronen der nächst höheren Schicht verbunden ist. Die Gewichte von dieser Verbindungen werden ganz normal gelernt und repräsentieren die Schwellen Θ der Nachfolgeneuronen.

9.5.1 NETtalk: ein Netz lernt das Sprechen

In beeindruckender Weise zeigten Sejnowski und Rosenberg 1986 auf, was Backpropagation zu leisten im Stande ist. [SR86] Sie bauten ein System, das in der Lage ist, einen in einer Textdatei gespeicherten englischen Text laut und verständlich vorzulesen. Die Architektur des in Abb. 9.21 dargestellten Netzes besteht aus einer Eingabeschicht mit $7 \times 29 = 203$

Neuronen in denen der aktuelle Buchstabe und drei Buchstaben davor sowie drei Buchstaben danach kodiert werden. Für jeden dieser sieben Buchstaben sind 29 Neuronen für „a . . z „, „.“ reserviert. Über 80 verdeckte Neuronen wird die Eingabe abgebildet auf die 26 Ausgabeneuronen, von denen jedes für einen bestimmten Laut steht. Zum Beispiel würde das „a“ in „father“ tief, betont, zentral ausgesprochen werden. Das Netz wurde trainiert mit 1000 Worten, die zufällig buchstabenweise nacheinander angelegt und trainiert wurden. Für jeden Buchstaben wurde als Zielausgabe manuell dessen Betonung angegeben. Um die Betonungsattribute in reale Töne umzusetzen, wurde ein Teil des Sprachsynthesysystems DECtalk verwendet. Durch eine vollständige Vernetzung erhält das Netz insgesamt $203 \times 80 + 80 \times 26 = 18.320$ Gewichte.

Auf einem Simulator auf einer VAX 780 wurde das System mit etwa 50 Zyklen über alle Worte trainiert. Bei etwa 5 Zeichen pro Wort im Mittel wurden also etwa $5 \cdot 50 \cdot 1000 = 250.000$ Iterationen des Backpropagation-Algorithmus benötigt. Bei einer Geschwindigkeit von etwa einem Zeichen pro Sekunde bedeutete das etwa 69 Stunden Rechenzeit. Die Entwickler beobachteten bei dem System viele Eigenschaften, die ganz ähnlich zu menschlichen Lernen sind. Am Anfang konnte das System undeutlich nur ganz einfache Worte sprechen. Im Laufe der Zeit wurde es immer besser und erreichte dann eine Korrektheit von 95 % der gesprochenen Buchstaben.

Für anschauliche Experimente mit neuronalen Netzen ist der Java Neural Network Simulator JNNS zu empfehlen [Zel94]. Das NETtalk Netzwerk, geladen und trainiert mit JNNS, ist in Abb. 9.22 dargestellt.

9.5.2 Lernen von Heuristiken für Theorembeweiser

In Kap. 6 wurden Verfahren zur heuristischen Suche, unter anderem der A^{*}-Algorithmus sowie der IDA^{*}-Algorithmus behandelt. Um eine deutliche Reduktion des Suchraums zu erreichen, wird eine gute Heuristik für die jeweilige Anwendung benötigt. In Abschn. 4.1 wurde das Problem der Suchraumexplosion bei der Suche von Theorembeweisern nach einem Beweis aufgezeigt. Die kombinatorische Explosion des Suchraums beim Beweisen wird erzeugt durch die in jedem Schritt große Zahl möglicher Inferenzschritte.

Man versucht nun, heuristische Beweissteuerungsmodule zu bauen, die die verschiedenen Alternativen für den nächsten Schritt bewerten und dann die Alternative mit der besten Bewertung auswählen. Im Fall der Resolution könnte die Bewertung der verfügbaren Klauseln über eine Funktion erfolgen, die aufgrund von bestimmten Attributen der Klauseln wie etwa die Zahl der Literale, die Zahl der positiven Literale, die Komplexität der Terme, etc. für jedes Paar von resolvierbaren Klauseln einen Wert berechnet. In [ESS89, SE90] wurde für den Theorembeweiser SETHEO mit Backpropagation eine solche Heuristik gelernt. Beim Lernen der Heuristik wurden für jeden gefundenen Beweis an jeder Verzweigung während der Suche die Attribute der aktuellen Klausel als Trainingsdatum für die positive Klasse gespeichert. An allen Verzweigungen, die nicht zum Beweis führen, wurden die Attribute der aktuellen Klausel als Trainingsdatum für die negative Klasse ge-

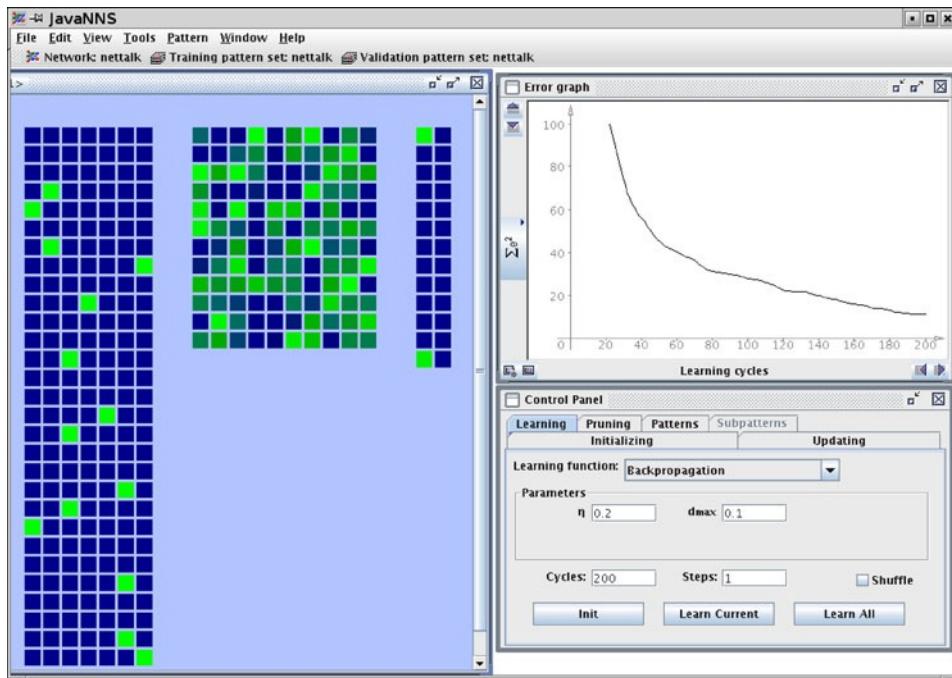


Abb. 9.22 Das NETtalk-Netzwerk in JNNS. Im linken Fenster von links nach rechts erkennt man die 7 · 29 Eingabeneuronen, 80 verdeckte und 26 Ausgabeneuronen. Aufgrund der großen Zahl sind die Gewichte im Netz ausgeblendet. Rechts oben ist eine Lernkurve mit der zeitlichen Entwicklung des quadratischen Fehlers zu sehen

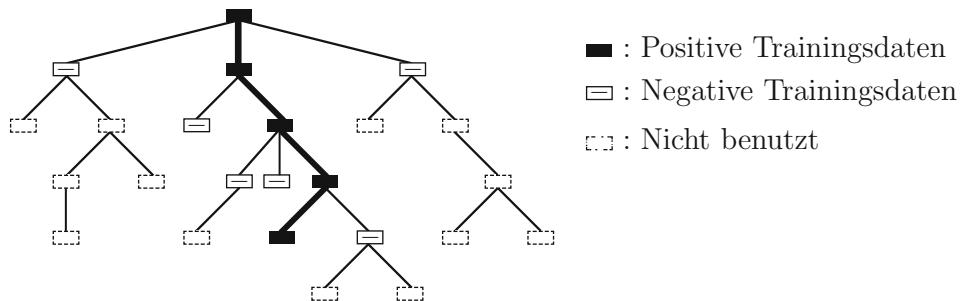


Abb. 9.23 Suchbaum mit einem erfolgreichen Pfad, dessen Knoten positiv bewertet werden und die negativ bewerteten nicht erfolgreichen Abzweigungen

speichert. Solch ein Baum mit einem erfolgreichen Pfad und entsprechenden Bewertungen der Knoten ist in Abb. 9.23 skizziert.

Auf diesen Daten wurde ein Backpropagation-Netzwerk trainiert und dann im Beweiser für die Bewertung der Klauseln verwendet. Es wurden 16 numerische Attribute für

jede Klausel berechnet, normiert und in je einem Eingabeneuron kodiert. Mit 25 verdeckten Neuronen und einem Ausgabeneuron für die Klasse (positiv/negativ) wurde das Netz trainiert.

Es konnte gezeigt werden, dass durch die gelernten Heuristiken die Zahl der versuchten Inferenzen bei schwierigen Problemen um viele Größenordnungen reduziert werden kann, was letztlich die Rechenzeiten von Stunden auf Sekunden reduzierte. Dadurch wurde es möglich, Sätze zu beweisen, die ohne Heuristik unerreichbar waren.

9.5.3 Probleme und Verbesserungen

Backpropagation ist mittlerweile über 20 Jahre alt und hat sich in den verschiedensten Anwendungen, zum Beispiel bei der Mustererkennung oder in der Robotik bewährt. Allerdings ist die Anwendung nicht immer unproblematisch. Insbesondere wenn das Netz viele Tausende Gewichte hat und viele Trainingsdaten gelernt werden sollen, treten zwei Probleme auf:

Oft konvergiert das Netz zu lokalen Minima der quadratischen Fehlerfunktion. Außerdem ist die Konvergenz von Backpropagation oft sehr langsam. Das heißt, es werden sehr viele Iterationen über alle Trainingsmuster benötigt. Um diese Probleme zu verringern, wurden mittlerweile viele Verbesserungen vorgeschlagen. Wie auch schon in Abschn. 9.4.3 erwähnt, können Oszillationen vermieden werden durch das in Abb. 9.15 dargestellte langsame Verkleinern der Lernrate η .

Eine andere Methode zur Verringerung von Oszillationen ist die Verwendung eines Impuls-Terms (engl. momentum) bei der Gewichtsänderung, der dafür sorgt, dass sich von einem Schritt zum nächsten die Richtung des Gradientenabstiegs nicht so dramatisch ändert. Hierbei wird zur aktuellen Gewichtsänderung $\Delta_p w_{ji}(t)$ zum Zeitpunkt t noch ein Teil der Änderung $\Delta_p w_{ji}(t-1)$ des Schritts davor addiert. Die Lernregel ändert sich dann zu

$$\Delta_p w_{ji}(t) = \eta \delta_j^p x_i^p + \gamma \Delta_p w_{ji}(t-1)$$

mit einem Parameter γ zwischen null und eins. An einem zweidimensionalen Beispiel ist dies in Abb. 9.24 veranschaulicht.

Eine weitere Idee führt zur Minimierung der linearen Fehlerfunktion statt der quadratischen, wodurch das Problem der langsamen Konvergenz in flachen Tälern reduziert wird.

Der Gradientenabstieg bei Backpropagation basiert letztlich auf einer linearen Näherung der Fehlerfunktion. Von Scott Fahlmann stammt Quickprop, ein Verfahren, das eine quadratische Näherung der Fehlerfunktion verwendet und damit schnellere Konvergenz erreicht.

Durch geschickte Vereinigung der erwähnten Verbesserungen und andere heuristische Tricks wurde von Martin Riedmiller mit dem Verfahren RProp eine weitere Optimierung erreicht [RB93]. In Abschn. 8.10 haben wir RProp zur Klassifikation der Appendizitisdaten angewendet und erreichen einen Fehler, der etwa gleich groß ist wie der eines gelernten Entscheidungsbaumes.

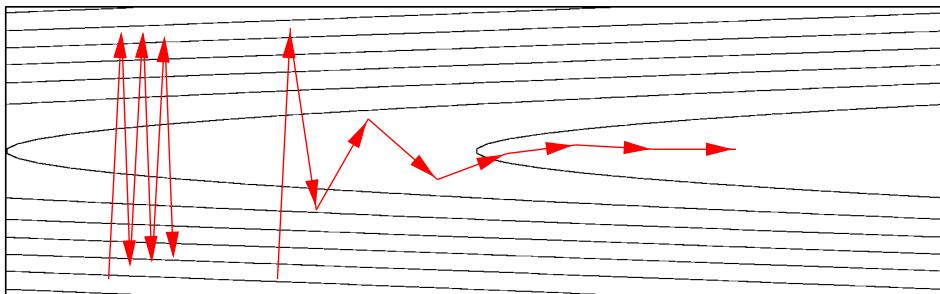


Abb. 9.24 Durch die Verwendung des Impulsterms werden abrupte Richtungsänderungen geglättet. Eine Iteration ohne Impuls-Term (*links*) im Vergleich zur Iteration mit Impuls-Term (*rechts*)

9.6 Support-Vektor-Maschinen

Gerichtete neuronale Netze mit nur einer Lage von Gewichten sind linear. Die Linearität führt zu einfachen Netzen und schnellem Lernen mit Konvergenzgarantie. Außerdem ist bei linearen Modellen die Gefahr des Overfitting gering. Für viele Anwendungen sind die linearen Modelle aber nicht stark genug, zum Beispiel weil die relevanten Klassen nicht linear separabel sind. Hier kommen mehrschichtige Netze wie zum Beispiel Backpropagation zum Einsatz mit der Konsequenz, dass lokale Minima, Konvergenzprobleme und Overfitting auftreten können.

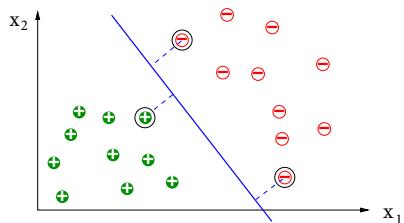
Ein vielversprechender Ansatz, die Vorteile der linearen und der nichtlinearen Modelle zu vereinen, wird durch die Theorie der Support-Vektor-Maschinen (SVM) verfolgt, die wir nun am Beispiel eines Zweiklassenproblems grob skizzieren wollen.⁴

Im Fall von zwei linear separablen Klassen ist es einfach, zum Beispiel mit der Perzeptron-Lernregel eine trennende Hyperebene zu finden. Allerdings gibt es meist unendlich viele solcher Ebenen, wie auch in dem zweidimensionalen Beispiel in Abb. 9.25. Gesucht ist nun eine Ebene, die zu beiden Klassen einen möglichst großen minimalen Abstand hat. Diese Ebene wird eindeutig definiert durch (meist) einige wenige Punkte im Grenzbereich. Diese Punkte, die so genannten **Support-Vektoren**, haben alle den gleichen Abstand zur Trenngeraden. Zum Finden der Support-Vektoren gibt es effiziente Optimierungsalgorithmen. Interessant ist nun, dass die optimale trennende Hyperebene durch einige wenige Parameter, nämlich durch die Supportvektoren, bestimmt wird. Damit ist die Gefahr der Überanpassung gering.

Bei den Support-Vektor-Maschinen wird nun versucht, dieses Verfahren auf nicht linear separable Probleme anzuwenden. Das Vorgehen ist ein zweistufiges: Im ersten Schritt wird auf die Daten eine nichtlineare Transformation angewendet mit der Eigenschaft, dass

⁴ Die Support-Vektor-Maschinen sind keine neuronalen Netze. Aufgrund der historischen Entwicklung und ihrer mathematischen Verwandtschaft zu linearen Netzen werden sie aber hier behandelt.

Abb. 9.25 Zwei Klassen mit der maximal trennenden Geraden. Die *umrandeten Punkte* sind die Support-Vektoren



die transformierten Daten linear separabel sind. Im zweiten Schritt werden dann im transformierten Raum die Support-Vektoren bestimmt.

Der erste Schritt ist hochinteressant, aber nicht ganz einfach. Zuerst stellen wir fest, dass es immer möglich ist, durch Transformation des Vektorraumes die Klassen linear separabel zu machen, sofern die Daten keine Widersprüche enthalten. Ein Datenpunkt ist widersprüchlich, wenn er zu beiden Klassen gehört. Solch eine Trennung erreicht man zum Beispiel durch Einführung einer neuen $(n+1)$ -ten Dimension und der Festlegung

$$x_{n+1} = \begin{cases} 1 & \text{falls } \mathbf{x} \in \text{Klasse 1} \\ 0 & \text{falls } \mathbf{x} \in \text{Klasse 0}. \end{cases}$$

Diese Formel hilft aber nicht viel weiter, denn sie ist auf neue zu klassifizierende Punkte mit unbekannter Klasse nicht anwendbar. Man benötigt also eine allgemeine Transformation, die möglichst unabhängig von den aktuellen Daten sein sollte. Man kann nun zeigen, dass es solche generischen Transformationen sogar für beliebig geformte Klassentrennlinien im ursprünglichen Vektorraum gibt. Im transformierten Raum sind die Daten dann linear separabel. Allerdings wächst dann die Zahl der Dimensionen des neuen Vektorraumes exponentiell mit der Zahl der Dimensionen des ursprünglichen Vektorraums. Die große Zahl neuer Dimensionen ist aber gar nicht so problematisch, denn bei Verwendung der Support-Vektoren wird die trennende Ebene, wie oben erwähnt, nur durch wenige Parameter bestimmt.

Die zentrale nichtlineare Transformation des Vektorraums wird als **Kernel** bezeichnet, weshalb die Support-Vektor-Maschinen auch unter dem Namen Kernel-Methoden bekannt sind. Die ursprünglich für Klassifikationsaufgaben entwickelte Theorie der SVMs wurde erweitert und ist nun auch auf Regressionsprobleme anwendbar.

Die hierbei verwendete Mathematik ist sehr interessant, aber zu aufwändig für eine erste Einführung. Für einen tieferen Einstieg in dieses vielversprechende junge Teilgebiet des maschinellen Lernens verweisen wir auf [SS02], [Alp04] oder [Bur98].

9.7 Anwendungen

Neben den hier gezeigten Anwendungsbeispielen für neuronale Netze gibt es heute in allen Industriebereichen unzählige Anwendungen für neuronale Netze. Ein ganz wichtiger Bereich ist die Mustererkennung in allen ihren Ausprägungen, sei es die Analyse von Fotos

zur Erkennung von Personen oder Gesichtern, das Erkennen von Fischschwärm auf Sonarechos, das Erkennen und Klassifizieren von militärischen Fahrzeugen auf Radarscans und viele mehr. Aber auch für das Erkennen von gesprochener Sprache und handschriftlichen Texten werden neuronale Netze trainiert.

Nicht nur für das Erkennen von Objekten und Szenen werden neuronale Netze verwendet. Auch auf das Steuern von einfachen Robotern, basierend auf Sensordaten, werden sie trainiert, genauso wie für die heuristische Steuerung der Suche in Backgammon- oder Schachcomputern. Interessant ist hierbei auch der in Abschn. 10.8 beschriebene Einsatz der Netze zum Lernen durch Verstärkung.

Schon seit längerer Zeit werden neben statistischen Methoden auch neuronale Netze erfolgreich zur Prognose von Aktienkursen oder zur Beurteilung der Kreditwürdigkeit von Bankkunden eingesetzt.

Für viele dieser Anwendungen können auch andere maschinelle Lernverfahren eingesetzt werden. Die neuronalen Netze waren aber in den Anwendungen bis heute deutlich erfolgreicher und populärer als alle anderen Verfahren des maschinellen Lernens. Dieses in der Vergangenheit große Übergewicht der neuronalen Netze verschiebt sich aber durch den großen kommerziellen Erfolg des Data Mining und auch durch die Support-Vektor-Maschinen langsam hin zu anderen Methoden.

9.8 Zusammenfassung und Ausblick

Mit dem Perzeptron, der Delta-Regel und Backpropagation haben wir die wichtigste Klasse neuronaler Netze eingeführt und deren Verwandtschaft zu den Scores und zu Naive-Bayes auf der einen Seite, aber auch zur Methode der kleinsten Quadrate aufgezeigt. Dem biologischen Vorbild am nächsten kommen vermutlich die faszinierenden Hopfield-Netze, die aber auf Grund ihrer komplexen Dynamik nur schwer in der Praxis handhabbar sind. Für die Praxis wichtiger sind aber die verschiedenen vorgestellten Assoziativspeichermodelle.

Bei allen neuronalen Modellen werden die Informationen verteilt über viele Gewichte gespeichert. Deswegen hat zum Beispiel das Absterben einiger weniger einzelner Neuronen keine merklichen Auswirkungen auf die Funktion eines Gehirns. Das Netzwerk ist durch die verteilte Speicherung der Daten robust gegenüber kleinen Störungen. Auch das in mehreren Beispielen gezeigte Wiedererkennen von fehlerhaften Mustern hängt damit zusammen.

Die verteilte Repräsentation des Wissens hat aber den Nachteil, dass es für den Wissensingenieur schwierig ist, Informationen zu lokalisieren. Es ist praktisch unmöglich, in einem fertig trainierten neuronalen Netz die vielen Gewichte zu analysieren und zu verstehen. Bei einem gelernten Entscheidungsbaum hingegen ist dies relativ einfach, das gelernte Wissen zu verstehen und es sogar als logische Formel darzustellen. Besonders ausdrucksstark und elegant ist die Prädikatenlogik, die es erlaubt, Relationen zu formalisieren. Zum Beispiel ein Prädikat *großmutter* (*katrin, klaus*) ist einfach zu verstehen. Auch mit neuronalen Netzen können derartige Relationen gelernt werden. Aber es ist eben nicht

möglich, das „Großmutter-Neuron“ im Netzwerk ausfindig zu machen. Daher gibt es bis heute immer noch Probleme bei der Verbindung von neuronalen Netzen mit symbolverarbeitenden Systemen.

Von der großen Zahl interessanter neuronaler Modelle konnten viele nicht behandelt werden. Sehr interessant sind zum Beispiel auch die von Kohonen eingeführten selbstorganisierenden Karten, eine biologisch motivierte Abbildung von einer sensorischen Neuronenschicht auf eine zweite Schicht von Neuronen mit der Eigenschaft, dass diese Abbildung adaptiv und Ähnlichkeitserhaltend ist.

Ein Problem bei den hier vorgestellten Netzen tritt beim inkrementellen Lernen auf. Wird zum Beispiel ein fertig trainiertes Backpropagation-Netz mit neuen Mustern weiter trainiert, so werden häufig viele, eventuell sogar alle alten Muster schnell vergessen. Um dieses Problem zu lösen entwickelten Carpenter und Grossberg die Adaptive Resonance Theory (ART), welche zu einer ganzen Reihe neuronaler Modelle führt.

Als weiterführende Literatur empfehlen wir die Lehrbücher [Bis05, Zel94, RMS91, Roj93]. Wer die wichtigsten Originalarbeiten in diesem spannenden Gebiet lesen will, dem seien die beiden Sammelbände [AR88, APR90] empfohlen.

9.9 Übungen

Von der Biologie zur Simulation

Aufgabe 9.1 Zeigen Sie die Punktsymmetrie der Sigmoidfunktion zu $(\theta, \frac{1}{2})$.

Hopfield-Netze

Aufgabe 9.2 Verwenden Sie das Hopfield-Netze-Applet auf <http://www.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html> und testen Sie bei Gittergröße 10×10 die Speicherkapazität bei korrelierten und bei unkorrelierten Mustern (mit zufällig gesetzten Bits). Verwenden Sie beim Testen Muster mit 10 % Rauschen.

Aufgabe 9.3 Vergleichen Sie den in Abschn. 9.2.2 angegebenen theoretischen Grenzwert $N = 0,146 n$ für die maximale Zahl speicherbarer Muster mit der Kapazität eines gleich großen klassischen binären Speichers.

Lineare Netze mit minimalem Fehler

Aufgabe 9.4 \Rightarrow

- Schreiben Sie ein Programm zum Lernen einer linearen Abbildung mit der Methode der kleinsten Quadrate. Dies ist ganz einfach: Es sind nur die Normalgleichungen nach (9.12) aufzustellen und dann das lineare Gleichungssystem zu lösen.

- b) Wenden Sie dieses Programm auf die Appendizitisdaten auf der Webseite zum Buch an und bestimmen Sie einen linearen Score. Geben Sie für diesen, wie in Abb. 9.14, den Fehler in Abhängigkeit von der frei wählbaren Schwelle.
- c) Bestimmen Sie nun die ROC-Kurve für diesen Score.

Backpropagation

Aufgabe 9.5 Verwenden Sie ein Backpropagation-Programm, zum Beispiel in JNNS oder in KNIME und kreieren Sie ein Netz mit acht Eingabeneuronen, acht Ausgabeneuronen und drei verdeckten Neuronen. Trainieren Sie nun das Netz mit acht Trainingsdatenpaaren $\mathbf{q}^p, \mathbf{q}^p$ mit der Eigenschaft, dass im p -ten Vektor \mathbf{q}^p das p -te Bit eins ist und alle anderen Bits null sind. Das Netz hat also die simple Aufgabe, eine identische Abbildung zu lernen. Solch ein Netz wird als 8-3-8 Encoder bezeichnet. Beobachten Sie die Kodierungen der drei verdeckten Neuronen nach dem Lernen bei Eingabe aller acht gelernten Eingabevektoren. Was fällt Ihnen auf? Reduzieren Sie nun die Zahl der verdeckten Neuronen auf zwei und eins und wiederholen das Experiment.

Aufgabe 9.6 Zeigen Sie, dass Backpropagation-Netze nicht linear separable Mengen trennen können, indem Sie die XOR-Funktion trainieren. Die Trainingsdaten hierfür sind: $((0, 0), 0), ((0, 1), 1), ((1, 0), 1), ((1, 1), 0)$.

- a) Verwenden Sie ein Netz mit je zwei Neuronen in der Eingabeschicht, und in der verdeckten Schicht und einem Ausgabeneuron und trainieren Sie es.
- b) Löschen Sie nun die verdeckten Neuronen und verbinden die Eingabeschicht direkt mit dem Ausgabeneuron. Was beobachten Sie?

Aufgabe 9.7

- a) Zeigen Sie, dass ein mehrlagiges Backpropagation-Netz mit linearer Aktivierungsfunktion gleich mächtig ist wie ein zweilagiges. Dazu genügt es zu zeigen, dass das hintereinander Ausführen von linearen Abbildungen eine lineare Abbildung ist.
- b) Zeigen Sie, dass ein zweilagiges Backpropagation-Netz mit einer streng monotonen Aktivierungsfunktion für Klassifikationsaufgaben nicht mächtiger ist als eines ohne beziehungsweise mit linearer Aktivierungsfunktion.

Support-Vektor-Maschinen

Aufgabe 9.8 * Gegeben seien zwei nicht linear separable zweidimensionale Mengen von Trainingsdaten M_+ und M_- . Alle Punkte in M_+ liegen innerhalb des Einheitskreises $x_1^2 + x_2^2 = 1$ und alle Punkte in M_- liegen außerhalb. Geben Sie eine Koordinatentransformation $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ an, welche die Daten linear separabel macht. Geben Sie auch die Gleichung der Trennerade an und skizzieren Sie die beiden Räume und die Datenpunkte.

10.1 Einführung

Alle bisher beschriebenen Lernverfahren arbeiten mit Lehrer. Sie gehören also zur Klasse des *Supervised Learning*. Beim Lernen mit Lehrer soll der Agent anhand von Trainingsdaten eine Abbildung der EingabevARIABLEN auf die AusgabevARIABLEN lernen. Wichtig ist hierbei, dass für jedes einzelne Trainingsbeispiel sowohl alle Werte der EingabevARIABLEN als auch alle Werte der AusgabevARIABLEN vorgegeben sind. Man braucht eben einen Lehrer, beziehungsweise eine Datenbank, in der die zu lernende Abbildung für genügend viele Eingabewerte näherungsweise definiert ist. Einzige Aufgabe des maschinellen Lernverfahrens ist es, das Rauschen aus den Daten herauszufiltern und eine Funktion zu finden, die auch zwischen den gegebenen Datenpunkten die gesuchte Abbildung gut approximiert.

Beim Lernen durch Verstärkung (engl. reinforcement learning) ist die Situation eine andere, ungleich schwierigere, denn hier sind keine Trainingsdaten verfügbar. Wir starten mit einem ganz einfachen anschaulichen Beispiel aus der Robotik, das dann zur Illustration der verschiedenen Verfahren dient.

Die Robotik ist das klassische Anwendungsfeld des Lernens durch Verstärkung, denn die Aufgaben für Roboter sind häufig derart komplex, dass sie sich zum Einen nicht einfach (durch Programme) kodieren lassen und zum Anderen keine Trainingsdaten verfügbar sind. Die Aufgabe des Roboters besteht darin, durch Versuch und Irrtum (oder Erfolg) herauszufinden, welche Aktionen in einer bestimmten Situation *gut* sind und welche nicht. In vielen Fällen lernen wir Menschen ganz ähnlich. Zum Beispiel wenn ein Kind das aufrechte Gehen lernt, erfolgt dies meist ohne Anleitung, einfach durch Verstärkung. Erfolgreiche Gehversuche werden belohnt durch das Vorwärtskommen. Erfolglose Versuche hingegen werden bestraft durch mehr oder weniger schmerzhafte Stürze. Auch in der Schule oder beim Erlernen vieler Sportarten sind positive und negative Verstärkung wichtige Faktoren des erfolgreichen Lernens (siehe Abb. 10.1).

Eine stark vereinfachte Fortbewegungsaufgabe soll im folgenden Beispiel gelernt werden.

Abb. 10.1 „Vielleicht sollte ich beim nächsten mal den Schwung etwas früher einleiten oder langsamer fahren?“ – Lernen durch negative Verstärkung



Beispiel 10.1

In Abb. 10.2 links ist ein Roboter dargestellt, dessen Mechanik nur aus einem quaderförmigen Klotz und einem Arm mit zwei Gelenken g_y und g_x besteht (siehe [KMK97]). Die einzigen möglichen Aktionen des Roboters sind die Bewegungen von g_y nach oben oder unten sowie von g_x nach links oder rechts. Wir erlauben außerdem die Bewegungen nur in festen diskreten Einheiten, zum Beispiel um 10 Grad. Die Aufgabe des Agenten besteht nun darin, eine Strategie (engl. *Policy*) zu erlernen, die es ihm erlaubt, sich möglichst schnell nach rechts zu bewegen. Ganz analog arbeitet der in Abb. 10.2 rechts dargestellte Laufroboter mit dem gleichen zweidimensionalen Zustandsraum.

Eine erfolgreiche Aktionsssequenz ist in Tab. 10.1 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 sind zu finden über [KMK97] und [Tok06].

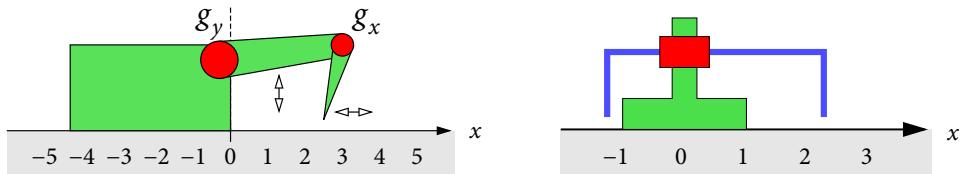


Abb. 10.2 Durch Bewegung der beiden Gelenke kann sich der Krabbelroboter *links im Bild* vorwärts und rückwärts bewegen. Der Laufroboter muss entsprechend den Rahmen auf und ab, bzw. links/rechts bewegen. Das Feedback für Bewegungen des Roboters nach rechts ist positiv und für Bewegungen nach links negativ

Tab. 10.1 Ein Zyklus einer periodischen Bewegungsfolge mit systematischer Vorwärtsbewegung

Krabbelroboter	Laufroboter	Zeit	Zustand	Belohnung	Aktion	
		t	g_y	g_x	x	a_t
		0	oben	links	0	rechts
		1	oben	rechts	0	tiefe
		2	unten	rechts	0	links
		3	unten	links	1	hoch

g_x	1	2	3	4	g_x	1	2	3	4
g_y	1				g_y	1			
ob.		2		3	ob.		2		3
unt.		3		4	unt.		4		

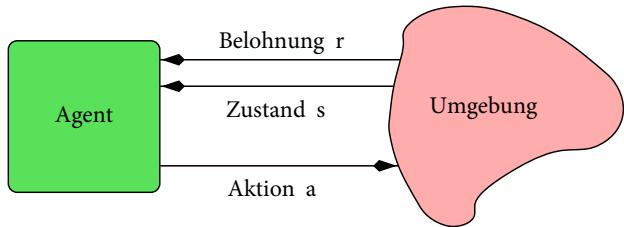
Abb. 10.3 Der Zustandsraum des Beispielroboters im Fall von je zwei möglichen Positionen der Gelenke (*links*) und im Fall von je vier horizontalen und vertikalen Positionen (*Mitte*). Im *rechten Bild* ist eine optimale Strategie angegeben

Bevor wir auf die Lernverfahren eingehen, müssen wir die Aufgabe angemessen mathematisch modellieren. Wir beschreiben zunächst den **Zustand** (engl. state) des Roboters durch die beiden Variablen g_x und g_y für die Stellungen der Gelenke mit jeweils endlich vielen diskreten Werten. Der Zustand des Roboters wird also beschrieben durch einen Vektor (g_x, g_y) . Die Zahl der möglichen Gelenkstellungen sei n_x , bzw. n_y . Zur Bewertung der Aktionen des Roboters verwenden wir die horizontale Position x des Roboterkörpers, die ganzzählige Werte annehmen kann. Bewegungen nach rechts werden mit positiven Änderungen von x belohnt und Bewegungen nach links mit negativen Änderungen bestraft.

In Abb. 10.3 ist der Zustandsraum für zwei Varianten dieses Beispiels vereinfacht¹ dargestellt. Im linken Beispiel haben beide Gelenke zwei und im mittleren Beispiel vier Stellungen. Rechts in Abb. 10.3 ist eine optimale Strategie angegeben.

¹ Der aus Kreissegmenten bestehende Raum der Armbewegungen ist durch ein rechteckiges Gitter wiedergegeben.

Abb. 10.4 Der Agent und seine Wechselwirkung mit der Umgebung



10.2 Die Aufgabenstellung

Wie in Abb. 10.4 dargestellt, unterscheiden wir zwischen dem Agenten und seiner Umgebung. Zum Zeitpunkt t wird die Welt, das heißt der Agent inklusive Umgebung, beschrieben durch einen **Zustand** $s_t \in \mathcal{S}$. Die Menge \mathcal{S} ist eine Abstraktion der tatsächlich möglichen Zustände der Welt, denn einerseits lässt sich die Welt nicht exakt beschreiben und andererseits hat der Agent, zum Beispiel wegen Messfehlern, oft nur unvollständige Informationen über den tatsächlichen Zustand. Der Agent führt dann zum Zeitpunkt t eine **Aktion** $a_t \in \mathcal{A}$ aus. Diese Aktion verändert die Welt und führt damit in den Zustand s_{t+1} zum Zeitpunkt $t+1$. Die durch die Umgebung definierte **Übergangsfunktion** δ bestimmt den neuen Zustand $s_{t+1} = \delta(s_t, a_t)$. Sie kann durch den Agenten nicht beeinflusst werden.

Nach ausgeführter Aktion a_t erhält der Agent ein Feedback r_t aus der Umgebung in Form einer **direkten Belohnung** (engl. immediate reward) $r_t = r(s_t, a_t)$ (siehe Abb. 10.4). Die direkte Belohnung $r_t = r(s_t, a_t)$ ist immer abhängig vom aktuellen Zustand und der ausgeführten Aktion. $r(s_t, a_t) = 0$ bedeutet, dass der Agent kein direktes Feedback für die Aktion a_t bekommt. Beim Lernen soll $r_t > 0$ zu einer positiven und $r_t < 0$ zu einer negativen Verstärkung der Bewertung der Aktion a_t im Zustand s_t führen. Beim Lernen durch Verstärkung werden insbesondere Anwendungen untersucht, bei denen über lange Zeit keine direkte Belohnung erfolgt. Ein Schachspieler lernt zum Beispiel aus gewonnenen oder verlorenen Partien seine Spielweise zu verbessern, auch wenn er für keinen einzelnen Zug eine direkte Belohnung erhält. Man erkennt hier gut die Schwierigkeit, die Belohnung am Ende einer Aktionssequenz den einzelnen Aktionen zuzuordnen (engl. credit assignment problem).

Im Fall des Laufroboters besteht der Zustand aus der Stellung der beiden Gelenke, also $s = (g_x, g_y)$. Die Belohnung ist gegeben durch den zurückgelegten Weg x .

Eine **Strategie** $\pi : \mathcal{S} \rightarrow \mathcal{A}$ ist eine Abbildung von Zuständen auf Aktionen. Ziel des Lernens durch Verstärkung ist es, dass der Agent basierend auf seinen Erfahrungen eine optimale Strategie lernt. Eine Strategie ist optimal, wenn sie langfristig, also über viele Schritte, die Belohnung maximiert. Unklar ist noch, was „Belohnung maximieren“ genau heißen soll. Dazu definieren wir den **Wert**, bzw. die **abgeschwächte Belohnung** (engl. discounted reward)

$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (10.1)$$

einer Strategie π wenn wir im Startzustand s_t starten. Hierbei ist $0 \leq \gamma < 1$ eine Konstante, die dafür sorgt, dass ein Feedback in der Zukunft umso stärker abgeschwächt wird, je weiter es in der Zukunft liegt. Die direkte Belohnung r_t wird am stärksten gewichtet. Diese Belohnungsfunktion wird überwiegend verwendet. Eine manchmal interessante Alternative ist die mittlere Belohnung

$$V^\pi(s_t) = \lim_{h \rightarrow \infty} \frac{1}{h} \sum_{i=0}^h r_{t+i}. \quad (10.2)$$

Eine Strategie π^* heißt optimal, wenn für alle Zustände s

$$V^{\pi^*}(s) \geq V^\pi(s) \quad (10.3)$$

gilt. Das heißt, sie ist mindestens so gut wie alle anderen Strategien entsprechend dem definierten Wertmaßstab. Zur besseren Lesbarkeit wird im Folgenden die optimale Wertfunktion V^{π^*} mit V^* bezeichnet.

Die hier behandelten Agenten beziehungsweise deren Strategien verwenden zur Bestimmung des nächsten Zustandes s_{t+1} nur Informationen über den aktuellen Zustand s_t und nicht über die Vorgeschichte. Dies ist gerechtfertigt, wenn die Belohnung einer Aktion nur von aktuellem Zustand und aktueller Aktion abhängt. Solche Prozesse nennt man **Markov-Entscheidungsprozesse** (engl. Markov decision process, MDP). In vielen Anwendungen, insbesondere in der Robotik, ist der tatsächliche Zustand des Agenten nicht exakt bekannt, was die Aktionsplanung weiter erschwert. Grund hierfür ist zum Beispiel ein verrausches Sensorsignal. Man bezeichnet den Prozess dann als **POMDP** (engl. partially observable Markov decision process).

10.3 Uninformierte kombinatorische Suche

Die einfachste Möglichkeit, eine erfolgreiche Strategie zu finden, ist das kombinatorische Aufzählen aller Strategien wie in Kap. 6 beschrieben. Schon bei dem einfachen Beispiel 10.1 gibt es jedoch sehr viele Strategien, was dazu führt, dass die kombinatorische Suche mit erheblichem Rechenaufwand verbunden ist. In Abb. 10.5 ist für jeden Zustand die Zahl der möglichen Aktionen angegeben. Daraus berechnet sich dann die Zahl der möglichen Strategien als das Produkt der angegebenen Werte, wie in Tab. 10.2 dargestellt.

Für beliebige Werte von n_x und n_y gibt es immer 4 Eckknoten mit 2 möglichen Aktionen, $2(n_x - 2) + 2(n_y - 2)$ Randknoten mit 3 Aktionen und $(n_x - 2)(n_y - 2)$ innere Knoten mit 4 Aktionen. Also gibt es

$$2^4 3^{2(n_x - 2) + 2(n_y - 2)} 4^{(n_x - 2)(n_y - 2)}$$

verschiedene Strategien bei festen n_x und n_y . Die Zahl der Strategien wächst also exponentiell mit der Zahl der Zustände. Dies gilt allgemein, falls es pro Zustand mehr als eine

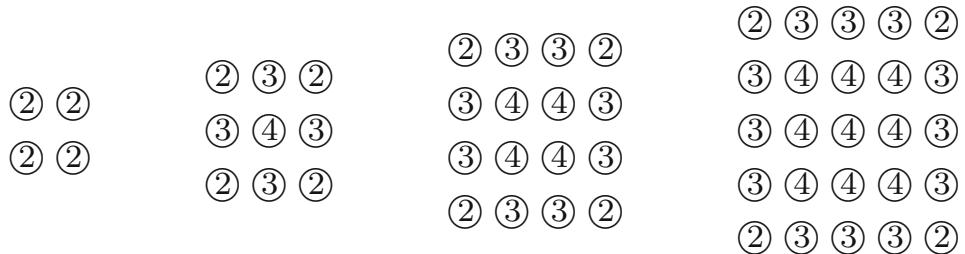


Abb. 10.5 Der Zustandsraum für das Beispiel mit den Werten 2, 3, 4, 5 für n_x und n_y . Zu jedem Zustand ist im Kreis die Zahl der möglichen Aktionen angegeben

Tab. 10.2 Zahl der Strategien für verschiedene große Zustandsräume im Beispiel

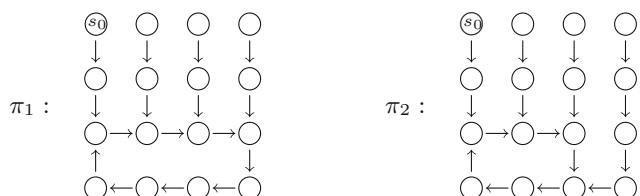
n_x, n_y	Anzahl d. Zustände	Anzahl d. Strategien
2	4	$2^4 = 16$
3	9	$2^4 3^4 = 5184$
4	16	$2^4 3^8 4^4 \approx 2,7 \cdot 10^7$
5	25	$2^4 3^{12} 4^9 \approx 2,2 \cdot 10^{12}$

mögliche Aktion gibt. Für praktische Anwendungen ist dieses Verfahren daher nicht tauglich. Auch die in Kap. 6 beschriebene heuristische Suche ist hier nicht anwendbar. Da die direkte Belohnung für fast alle Aktionen null ist, lässt sie sich nicht als heuristische Bewertungsfunktion verwenden.

Zum Finden einer optimalen Strategie muss neben dem Aufzählen aller Strategien für jede erzeugte Strategie π und jeden Startzustand s der Wert $V^\pi(s)$ berechnet werden, was den Aufwand noch weiter erhöht. Die unendliche Summe in $V^\pi(s)$ muss bei der praktischen Berechnung abgebrochen werden, was aber wegen der exponentiellen Abnahme der γ^i -Faktoren in (10.1) kein Problem darstellt.

In Beispiel 10.1 kann für eine Aktion a_t die Differenz $x_{t+1} - x_t$ als direkte Belohnung verwendet werden, was bedeutet, dass jede Bewegung des Roboterkörpers nach rechts mit 1 belohnt und jede Bewegung nach links mit -1 bestraft wird. In Abb. 10.6 sind zwei Strategien dargestellt. Hier ist die direkte Belohnung überall gleich null, außer in der untersten Zeile des Zustandsraumes. Die linke Strategie π_1 ist langfristig gesehen die bessere, denn

Abb. 10.6 Zwei verschiedene Strategien für das Beispiel



für lange Aktionssequenzen ist der mittlere Vortrieb pro Aktion $3/8 = 0,375$ für π_1 und $2/6 \approx 0,333$ für die rechte Strategie π_2 . Verwenden wir (10.1) für $V^\pi(s)$, so ergibt sich beim Startzustand s_0 links oben mit verschiedenen γ -Werten folgende Tabelle

γ	0,9	0,8375	0,8
$V^{\pi_1}(s_0)$	2,52	1,156	0,77
$V^{\pi_2}(s_0)$	2,39	1,156	0,80

die zeigt, dass für $\gamma = 0,9$ die längere Strategie π_1 besser abschneidet und für $\gamma = 0,8$ hingegen die kürzere Strategie π_2 . Bei $\gamma \approx 0,8375$ sind beide Strategien gleich gut. Man erkennt gut, dass ein größeres γ einen größeren Zeithorizont für die Bewertung von Strategien zur Folge hat.

10.4 Wert-Iteration und Dynamische Programmierung

Bei der naiven Vorgehensweise des Aufzählens aller Strategien wird viel redundante Arbeit geleistet, denn viele Strategien sind zu großen Teilen identisch. Sie unterscheiden sich eventuell nur minimal. Trotzdem wird jede Strategie komplett neu erzeugt und bewertet. Es bietet sich an, Zwischenergebnisse über Teile von Strategien zu speichern und wiederzuverwenden. Diese Vorgehensweise zur Lösung von Optimierungsproblemen wurde unter dem Namen **Dynamische Programmierung** von Richard Bellman schon 1957 beschrieben [Bel57]. Bellman erkannte, dass für eine optimale Strategie gilt:

Unabhängig vom Startzustand s_t und der ersten Aktion a_t müssen ausgehend von jedem möglichen Nachfolgezustand s_{t+1} alle folgenden Entscheidungen optimal sein.

Basierend auf dem so genannten Bellman-Prinzip wird es möglich, eine global optimale Strategie durch lokale Optimierungen einzelner Aktionen zu finden. Für MDPs werden wir nun dieses Prinzip zusammen mit einem geeigneten Iterationsverfahren herleiten.

Gesucht ist eine optimale Strategie π^* , welche (10.3) und (10.1) erfüllt. Wir schreiben die beiden Gleichungen um und erhalten

$$V^*(s_t) = \max_{a_t, a_{t+1}, a_{t+2}, \dots} (r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots). \quad (10.4)$$

Da die direkte Belohnung $r(s_t, a_t)$ nur von s_t und a_t abhängt, aber nicht von den Nachfolgezuständen und Aktionen, kann die Maximierung aufgeteilt werden, was letztlich zu folgender rekursiven Charakterisierung von V^* führt:

$$V^*(s_t) = \max_{a_t} [r(s_t, a_t) + \gamma \max_{a_{t+1}, a_{t+2}, \dots} (r(s_{t+1}, a_{t+1}) + \gamma r(s_{t+2}, a_{t+2}) + \dots)] \quad (10.5)$$

$$= \max_{a_t} [r(s_t, a_t) + \gamma V^*(s_{t+1})]. \quad (10.6)$$

Abb. 10.7 Der Algorithmus für die Wert-Iteration

```

WERT-ITERATION()
For all  $s \in \mathcal{S}$ 
     $\hat{V}(s) = 0$ 
Repeat
    For all  $s \in \mathcal{S}$ 
         $\hat{V}(s) = \max_a[r(s, a) + \gamma \hat{V}(\delta(s, a))]$ 
    Until  $\hat{V}(s)$  sich nicht mehr ändert

```

Gleichung (10.6) ergibt sich durch die Ersetzung $t \rightarrow t + 1$ in (10.4). Noch etwas einfacher geschrieben ergibt sich

$$V^*(s) = \max_a[r(s, a) + \gamma V^*(\delta(s, a))]. \quad (10.7)$$

Diese Gleichung besagt, wie auch (10.1), dass zur Berechnung von $V^*(s)$ die direkte Belohnung zu den mit dem Faktor γ abgeschwächten Belohnungen aller Nachfolgezustände addiert wird. Ist $V^*(\delta(s, a))$ bekannt, so ergibt sich $V^*(s)$ offenbar durch eine einfache lokale Optimierung über alle im Zustand s möglichen Aktionen a . Dies entspricht dem Bellman-Prinzip, weshalb (10.7) auch als Bellman-Gleichung bezeichnet wird.

Die optimale Strategie $\pi^*(s)$ führt im Zustand s eine Aktion aus, die zum maximalen Wert V^* führt. Also gilt

$$\pi^*(s) = \operatorname{argmax}_a[r(s, a) + \gamma V^*(\delta(s, a))]. \quad (10.8)$$

Aus der Rekursionsgleichung (10.7) ergibt sich nun in naheliegender Weise eine Iterationsvorschrift zur Berechnung von V^* :

$$\hat{V}(s) = \max_a[r(s, a) + \gamma \hat{V}(\delta(s, a))]. \quad (10.9)$$

Zu Beginn werden die Näherungswerte $\hat{V}(s)$ für alle Zustände initialisiert, zum Beispiel mit dem Wert null. Wiederholt wird nun $\hat{V}(s)$ für jeden Zustand aktualisiert, indem rekursiv über (10.9) auf den Wert $\hat{V}(\delta(s, a))$ des besten Nachfolgezustandes zurückgegriffen wird. Dieses Verfahren zur Berechnung von V^* heißt **Wert-Iteration** (engl. value iteration) und ist in Abb. 10.7 schematisch dargestellt. Man kann zeigen, dass die Wert-Iteration gegen V^* konvergiert [SB98]. Eine exzellente Analyse von Algorithmen zur dynamischen Programmierung liefert Szepesvari in [Sze10], wo, basierend auf Kontraktionseigenschaften der Algorithmen (zum Beispiel Wertiteration) mit Hilfe des Banachschen Fixpunktsatzes die Konvergenz bewiesen wird.

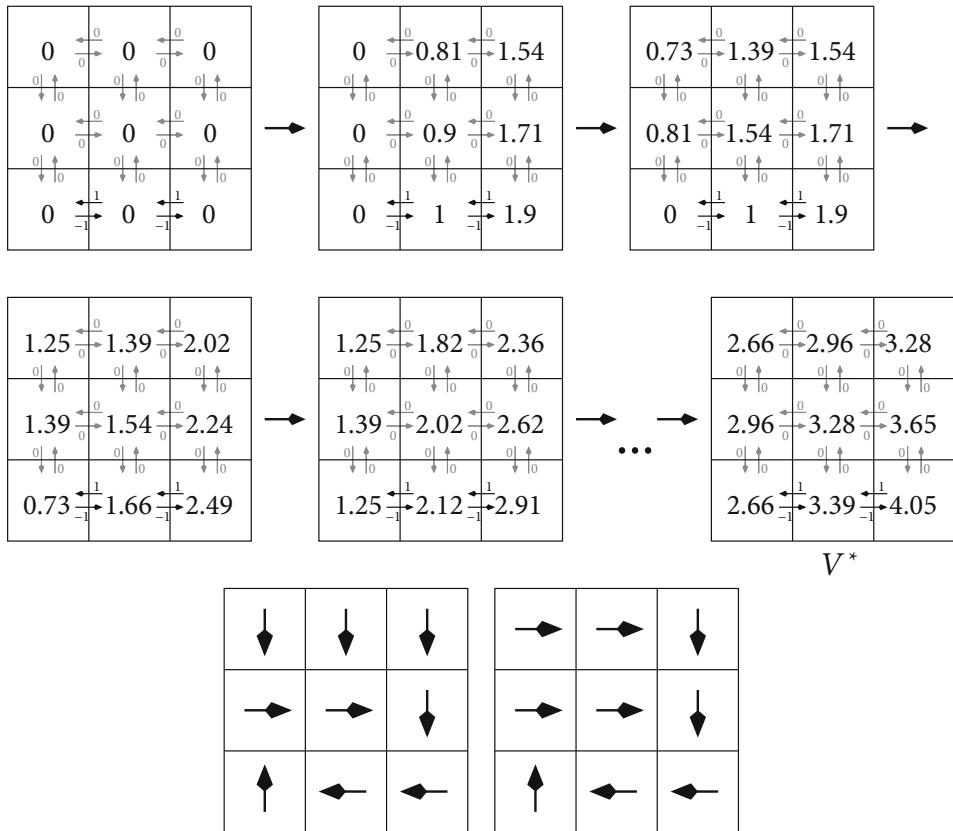


Abb. 10.8 Wert-Iteration am Beispiel mit 3×3 Zuständen. Die *letzten beiden Bilder* zeigen zwei optimale Strategien. Die Zahlen neben den Pfeilen geben die direkte Belohnung $r(s, a)$ der jeweiligen Aktion an

In Abb. 10.8 ist dieser Algorithmus auf das Beispiel 10.1 mit $\gamma = 0.9$ angewendet. In jeder Iteration werden die Zustände zeilenweise von links unten nach rechts oben abgearbeitet. Angegeben sind einige Anfangsiterationen und die stabilen Grenzwerte für V^* im zweiten Bild der unteren Reihe. Rechts daneben sind zwei optimale Strategien dargestellt.

Man erkennt in dieser Sequenz gut den Ablauf des Lernens. Der Agent exploriert wiederholt alle Zustände, führt für jeden Zustand die Wert-Iteration aus und speichert die Strategie in Form einer tabellierten Funktion V^* , welche dann noch in eine effizienter nutzbare Tabelle π^* kompiliert werden kann.

Zum Finden einer optimalen Strategie aus V^* wäre es übrigens falsch, im Zustand s_t die Aktion zu wählen, welche zum Zustand mit maximalem V^* -Wert führt. Entsprechend (10.8) muss noch die direkte Belohnung $r(s_t, a_t)$ addiert werden, denn gesucht ist $V^*(s_t)$

und nicht $V^*(s_{t+1})$. Angewendet auf den Zustand $s = (2, 3)$ in Abb. 10.8 bedeutet dies

$$\begin{aligned}\pi^*(2, 3) &= \underset{a \in \{\text{links, rechts, oben}\}}{\operatorname{argmax}} [r(s, a) + \gamma V^*(\delta(s, a))] \\ &= \underset{\{\text{links, rechts, oben}\}}{\operatorname{argmax}} \{1 + 0,9 \cdot 2,66, -1 + 0,9 \cdot 4,05, 0 + 0,9 \cdot 3,28\} \\ &= \underset{\{\text{links, rechts, oben}\}}{\operatorname{argmax}} \{3,39, 2,65, 2,95\} = \text{links.}\end{aligned}$$

An (10.8) erkennt man, dass der Agent im Zustand s_t für die Auswahl der optimalen Aktion a_t die direkte Belohnung r_t und den Nachfolgezustand $s_{t+1} = \delta(s_t, a_t)$ kennen muss. Er muss also ein Modell der Funktionen r und δ besitzen. Da dies jedoch für die meisten praktischen Anwendungen nicht zutrifft, werden Verfahren benötigt, die auch ohne die Kenntnis von r und δ arbeiten. Der Abschn. 10.6 ist solchen Verfahren gewidmet.

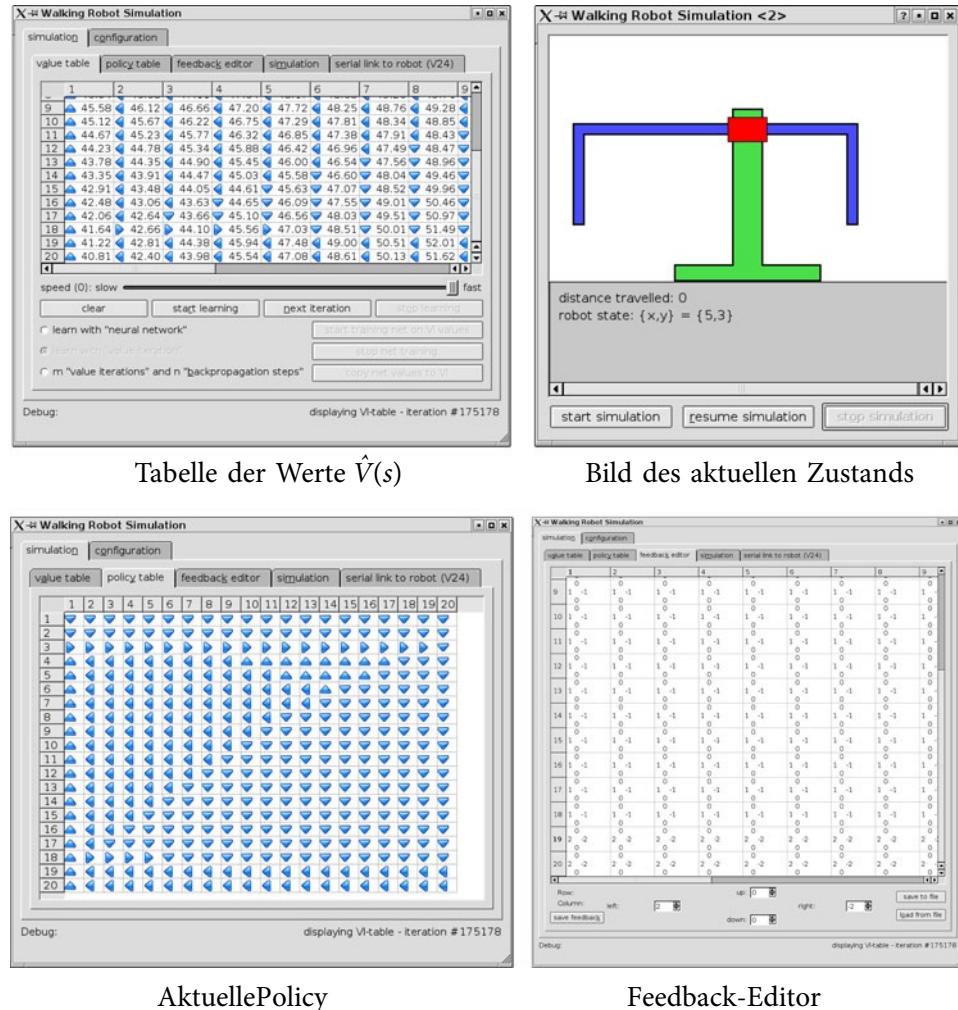
10.5 Ein lernender Laufroboter und seine Simulation

Eine graphische Benutzeroberfläche für einfache Experimente mit Lernen durch Verstärkung ist in Abb. 10.9 dargestellt [TEF09]. Der Benutzer kann für unterschiedlich große zweidimensionale Zustandsräume das Lernen durch Verstärkung beobachten. Zur besseren Generalisierung kann ein Backpropagation-Netz verwendet werden, welches die Zustände speichert (siehe hierzu Abschn. 10.8). Besonders interessant für Experimente ist der rechts unten dargestellte Feedback-Editor, mit dem der Nutzer selbst das Feedback der Umgebung vorgeben kann. Nicht dargestellt ist das Menu zur Einstellung der Parameter für die Wert-Iteration und das Lernen mit Backpropagation.

Neben der Simulation wurden speziell für die Lehre zwei kleine reale Krabbelroboter mit dem gleichen zweidimensionalen diskreten Zustandsraum entwickelt [TEF09].² In Abb. 10.10 sind die beiden Roboter dargestellt, die über jeweils zwei Modellbauservos bewegt werden. Angesteuert werden die Servos über einen Microcontroller oder über eine Funkschnittstelle direkt vom PC. Mit Hilfe der Simulationssoftware kann über die Funkschnittstelle zum Beispiel die Feedbackmatrix des Roboters auf dem PC visualisiert werden. Mit diesem gespeicherten Feedback kann dann auf dem schnelleren PC eine Strategie trainiert werden, die wiederum auf den Roboter geladen und ausgeführt wird. Der Roboter kann aber auch autonom lernen. Bei einem Zustandsraum der Größe 5×5 dauert dies etwa 30 Sekunden.

Interessant zu beobachten ist der Unterschied zwischen der Simulation und dem „richtigen“ Roboter. Im Gegensatz zur Simulation lernt der Krabbler zum Beispiel Strategien, bei denen er seinen Arm nie vom Boden abhebt, sich aber recht effizient fortbewegt. Grund ist der kleine asymmetrische Metallhaken an der Spitze des „Unterarms“, der je nach Un-

² Weitere Informationen und Bezugsquellen zum Krabbelroboter sind über <http://www.hs-weingarten.de/~ertel/kibuch> verfügbar.

**Abb. 10.9** Vier verschiedene Fenster des Laufroboter-Simulators

tergrund bei der Rückwärtsbewegung des Arms greift, aber bei der Vorwärtsbewegung durchrutscht. Diese Effekte werden über die Wegmesssensoren sehr sensibel wahrgenommen und beim Lernen entsprechend ausgewertet.

Die Adaptivität des Roboters führt zu überraschenden Effekten. Zum Beispiel konnten wir beobachten, wie der Krabbler mit einem defekten Servo, der in einem Winkelbereich durchrutschte, trotzdem noch das Laufen (eher Humpeln) lernte. Er ist eben in der Lage, sich durch eine geänderte Strategie an die veränderte Situation anzupassen. Ein durchaus erwünschter Effekt ist die Fähigkeit, bei unterschiedlich glattem Untergrund (zum Beispiel verschiedenen groben Teppichen) eine jeweils optimale Strategie zu lernen. Auch zeigt sich, dass



Abb. 10.10 Der Krabbelroboter in zwei Versionen

der reale Roboter schon mit einem kleinen Zustandsraum der Größe 5×5 sehr anpassungsfähig ist.

Der Leser möge (mangels richtigem Roboter) in der Simulation durch Variation der Feedback-Werte verschiedene Untergründe, beziehungsweise Defekte der Servos modellieren und dann die resultierende Strategie beobachten (Aufgabe 10.3).

10.6 Q-Lernen

Eine Strategie basierend auf der Bewertung von möglichen Nachfolgezuständen ist offenbar nicht anwendbar, wenn der Agent kein Modell der Welt hat, das heißt wenn er nicht weiß, in welchen Zustand ihn eine mögliche Aktion führt. Bei den meisten realistischen Anwendungen kann der Agent nicht auf solch ein Modell der Welt zurückgreifen. Zum Beispiel ein Roboter, der komplexe Objekte greifen soll, kann in vielen Fällen nicht vorhersagen, ob nach einer Greifaktion das zu greifende Objekt fest in seinem Greifer sitzt oder noch an seinem Platz liegt.

Bei fehlendem Modell der Welt wird eine Bewertung einer im Zustand s_t ausgeführten Aktion a_t benötigt, auch wenn noch unbekannt ist, wohin diese Aktion führt. Also arbeiten wir nun mit einer Bewertungsfunktion $Q(s_t, a_t)$ für Zustände mit zugehöriger Aktion. Mit dieser Funktion erfolgt die Auswahl der optimalen Aktion nach der Vorschrift

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a). \quad (10.10)$$

Zur Definition der Bewertungsfunktion verwenden wir wieder das schrittweise Abschwächen der Bewertung für weiter weg liegende zukünftige Zustands-Aktions-Paare genau wie in (10.1). Wir wollen also $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$ maximieren und definieren daher zur Bewertung der Aktion a_t im Zustand s_t

$$Q(s_t, a_t) = \max_{a_{t+1}, a_{t+2}, \dots} (r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots). \quad (10.11)$$

Analog zum Vorgehen bei der Wert-Iteration bringen wir diese Gleichung durch

$$Q(s_t, a_t) = \max_{a_{t+1}, a_{t+2}, \dots} (r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots) \quad (10.12)$$

$$= r(s_t, a_t) + \gamma \max_{a_{t+1}, a_{t+2}, \dots} (r(s_{t+1}, a_{t+1}) + \gamma r(s_{t+2}, a_{t+2}) + \dots) \quad (10.13)$$

$$= r(s_t, a_t) + \gamma \max_{a_{t+1}} (r(s_{t+1}, a_{t+1}) + \gamma \max_{a_{t+2}} (r(s_{t+2}, a_{t+2}) + \dots)) \quad (10.14)$$

$$= r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \quad (10.15)$$

$$= r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(\delta(s_t, a_t), a_{t+1}) \quad (10.16)$$

$$= r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \quad (10.17)$$

in eine einfache rekursive Form.

Was haben wir im Vergleich zur Wert-Iteration nun gewonnen? Eigentlich wurden die alten Gleichungen nur leicht umgeschrieben. Aber genau hier liegt der Ansatz zu einem neuen Verfahren. Statt V^* zu speichern wird nun die Funktion Q gespeichert, und der Agent kann ohne ein Modell der Funktionen δ und r seine Aktionen auswählen. Es fehlt also nur noch ein Verfahren, um Q direkt, das heißt ohne Kenntnis von V^* , zu lernen.

Aus der rekursiven Formulierung für $Q(s, a)$ lässt sich in naheliegender Weise ein Iterationsverfahren zur Bestimmung von $Q(s, a)$ ableiten. Wir initialisieren eine Tabelle $\hat{Q}(s, a)$ für alle Zustände beliebig, zum Beispiel mit Nullen, und führen dann iterativ

$$\hat{Q}(s, a) = r(s, a) + \gamma \max_{a'} \hat{Q}(\delta(s, a), a') \quad (10.18)$$

aus. Hierbei ist noch zu bemerken, dass wir die Funktionen r und δ nicht kennen. Gelöst wird dieses Problem ganz pragmatisch dadurch, dass wir den Agenten in seiner Umgebung im Zustand s eine Aktion a ausführen lassen. Der Nachfolgezustand ist dann offenbar $\delta(s, a)$ und die Belohnung erhält der Agent von der Umgebung. Der in Abb. 10.11 dargestellte Algorithmus implementiert dieses Verfahren zum Q-Lernen.

Die Anwendung des Verfahrens auf Beispiel 10.1 mit $\gamma = 0,9$ und $n_x = 3, n_y = 2$, das heißt im 2×3 -Gitter, ist beispielhaft dargestellt in Abb. 10.12. Im ersten Bild sind alle Q -Werte auf null initialisiert. Im zweiten Bild, nach der ersten Aktionssequenz, werden die vier r -Werte, die ungleich null sind, als Q -Werte sichtbar. Im letzten Bild ist die gelernte optimale Strategie angegeben. Dass dieses Verfahren nicht nur im Beispiel, sondern allgemein konvergiert, zeigt der folgende Satz, dessen Beweis in [Mit97] zu finden ist.

Satz 10.1

Gegeben sei ein deterministischer MDP mit beschränkten direkten Belohnungen $r(s, a)$. Zum Lernen wird (10.18) mit $0 \leq \gamma < 1$ verwendet. Sei $\hat{Q}_n(s, a)$ der Wert für $\hat{Q}(s, a)$ nach n Aktualisierungen. Wird jedes Zustands-Aktions-Paar unendlich oft besucht, so konvergiert $\hat{Q}_n(s, a)$ für alle Werte von s und a gegen $Q(s, a)$ für $n \rightarrow \infty$.

Q-LERNEN()

For all $s \in S, a \in \mathcal{A}$

$$\hat{Q}(s, a) = 0 \text{ (oder zufällig)}$$

Repeat

Wähle (z.B. zufällig) einen Zustand s

Repeat

Wähle eine Aktion a und führe sie aus

Erhalte Belohnung r und neuen Zustand s'

$$\hat{Q}(s, a) := r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

$s := s$

Until s ist ein Endzustand **Oder** Zeitschranke erreicht

Until \hat{Q} konvergiert

Abb. 10.11 Der Algorithmus für das Q-Lernen

Beweis Da jeder Zustands-Aktions-Übergang unendlich oft vorkommt, betrachten wir aufeinanderfolgende Zeitintervalle mit der Eigenschaft, dass in jedem Intervall alle Zustands-Aktions-Übergänge mindestens einmal vorkommen. Wir zeigen nun, dass der maximale Fehler über alle Einträge in der \hat{Q} -Tabelle in jedem dieser Intervalle um mindestens den Faktor γ reduziert wird. Sei

$$\Delta_n = \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

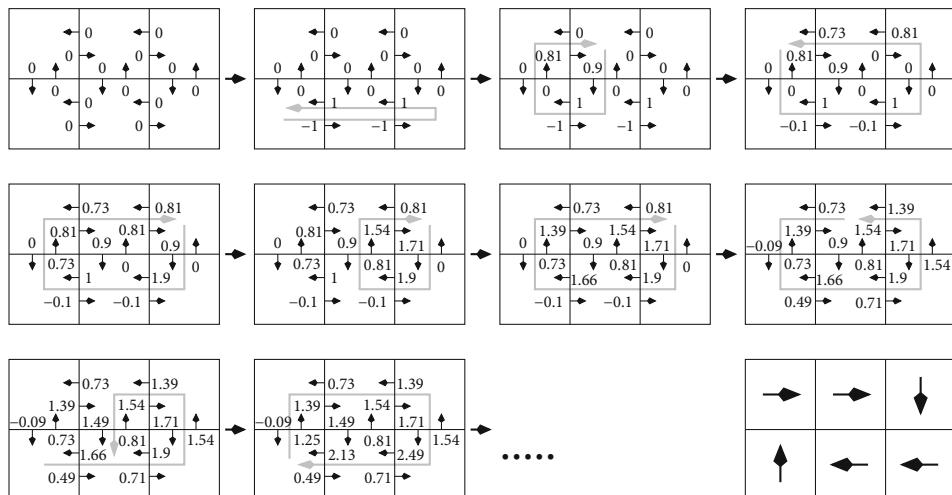


Abb. 10.12 Q-Lernen angewandt auf das Beispiel mit $n_x = 3, n_y = 2$. Die grauen Pfeile markieren die im jeweiligen Bild ausgeführten Aktionen. Die aktualisierten Q-Werte sind angegeben. Im letzten Bild ist die aktuelle und zugleich optimale Strategie dargestellt

der maximale Fehler in der Tabelle \hat{Q}_n und $s' = \delta(s, a)$. Nun berechnen wir für jeden Tabelleneintrag $\hat{Q}_n(s, a)$ den Betrag des Fehlers nach einem Intervall zu

$$\begin{aligned} |\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\ &\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| = \gamma \Delta_n. \end{aligned}$$

Die erste Ungleichung gilt, weil für beliebige Funktionen f und g

$$|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|$$

und die zweite, weil durch zusätzliches Variieren des Zustandes s'' das resultierende Maximum nicht kleiner werden kann. Nun ist also gezeigt, dass $\Delta_{n+1} \leq \gamma \Delta_n$. Da der Fehler in jedem Intervall um mindestens den Faktor γ reduziert wird, ist er nach k Intervallen höchstens $\gamma^k \Delta_0$ und Δ_0 ist beschränkt. Da jeder Zustand unendlich oft besucht wird, gibt es unendlich viele Intervalle und Δ_n konvergiert gegen null. \square

Das Q-Lernen konvergiert nach Satz 10.1 offenbar unabhängig von den während des Lernens gewählten Aktionen. Das heißt, dass es für die Konvergenz keine Rolle spielt, wie der Agent seine Aktionen auswählt, solange jedes Zustands-Aktions-Paar unendlich oft besucht wird. Die Konvergenzgeschwindigkeit hingegen hängt sehr wohl davon ab, auf welchen Wegen der Agent sich während des Lernens bewegt (siehe Abschn. 10.7).

10.6.1 Q-Lernen in nichtdeterministischer Umgebung

In vielen Robotik-Anwendungen ist die Umgebung des Agenten nichtdeterministisch. Das heißt, die Reaktion der Umgebung auf die Aktion a im Zustand s zu zwei verschiedenen Zeitpunkten kann zu unterschiedlichen Nachfolgezuständen und Belohnungen führen. Modelliert wird solch ein nichtdeterministischer Markov-Prozess durch eine probabilistische Übergangsfunktion $\delta(s, a)$ und direkte Belohnung $r(s, a)$. Zur Definition der Q-Funktion muss dann jeweils der Erwartungswert über alle möglichen Nachfolgezustände berechnet werden. Die (10.17) wird also verallgemeinert zu

$$Q(s_t, a_t) = E(r(s, a)) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a'), \quad (10.19)$$

wobei $P(s' | s, a)$ die Wahrscheinlichkeit ist, im Zustand s mit Aktion a in den Nachfolgezustand s' zu gelangen. Leider gibt es im nichtdeterministischen Fall keine Konvergenzgarantie für das Q-Lernen, wenn wir wie bisher nach (10.18) vorgehen, denn in aufeinanderfolgenden Durchläufen der äußeren Schleife des Algorithmus in Abb. 10.11 können

Belohnung und Nachfolgezustand bei gleichem Zustand s und gleicher Aktion a völlig verschieden sein, was zu einer alternierenden Folge führt, die zum Beispiel zwischen mehreren Werten hin und her springt. Um derartige, stark springende Q -Werte zu vermeiden, addieren wir zur rechten Seite aus (10.18) den alten gewichteten Q -Wert nochmal dazu. Dies wirkt stabilisierend auf die Iteration. Die Lernregel lautet dann

$$\hat{Q}_n(s, a) = (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n[r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(\delta(s, a), a')] \quad (10.20)$$

mit einem zeitlich variablen Gewichtungsfaktor

$$\alpha_n = \frac{1}{1 + b_n(s, a)}.$$

Der Wert $b_n(s, a)$ gibt an, wie oft bis zur n -ten Iteration im Zustand s die Aktion a schon ausgeführt wurde. Für kleine Werte von b_n , das heißt am Anfang des Lernens, kommt der stabilisierende Term $\hat{Q}_{n-1}(s, a)$ nicht zum Tragen, denn der Lernprozess soll schnelle Fortschritte machen. Später jedoch erhält dieser Term immer mehr Gewicht und verhindert dadurch zu große Sprünge in der Folge der \hat{Q} -Werte. Beim Einbau von (10.20) in das Q-Lernen müssen die Werte $b_n(s, a)$ für alle Zustands-Aktions-Paare gespeichert werden. Es bietet sich an, dies über eine Erweiterung der Tabelle für die \hat{Q} -Werte zu realisieren.

Zum besseren Verständnis von (10.20) vereinfachen wir diese, indem wir $\alpha_n = \alpha$ als konstant annehmen und wie folgt umformen:

$$\begin{aligned} \hat{Q}_n(s, a) &= (1 - \alpha)\hat{Q}_{n-1}(s, a) + \alpha[r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(\delta(s, a), a')] \\ &= \hat{Q}_{n-1}(s, a) + \underbrace{\alpha [r(s, a) + \gamma \max_{a'} \hat{Q}_{n-1}(\delta(s, a), a') - \hat{Q}_{n-1}(s, a)]}_{\text{TD-Fehler}} \end{aligned}$$

Der neue Q -Wert $\hat{Q}_n(s, a)$ lässt sich offenbar darstellen als der alte $\hat{Q}_{n-1}(s, a)$ plus einen Korrekturterm, der gleich der Änderung des Q -Werts in diesem Schritt ist. Der Korrekturterm wird als TD-Fehler (engl. TD-error, bzw. temporal difference error) bezeichnet und die obige Gleichung zur Veränderung des Q -Werts ist ein Spezialfall des TD-Lernens, einer wichtigen Klasse von Lernverfahren [SB98]. Für $\alpha = 1$ erhält man das oben beschriebene Q-Lernen. Für $\alpha = 0$ werden die \hat{Q} -Werte gar nicht geändert. Es findet also kein Lernen statt.

10.7 Erkunden und Verwerten

Für das Q-Lernen wurde bisher nur ein grobes Algorithmen-Schema angegeben. Es fehlt insbesondere eine Beschreibung der Auswahl des jeweiligen Startzustandes und der auszuführenden Aktion in der inneren Schleife von Abb. 10.11. Für die Auswahl der nächsten

Aktion gibt es prinzipiell zwei Möglichkeiten. Unter den möglichen Aktionen kann zufällig eine ausgewählt werden. Dies führt langfristig zu einem gleichmäßigen Erkunden (engl. exploration) aller möglichen Aktionen, beziehungsweise Strategien, aber mit sehr langsamer Konvergenz. Alternativ hierzu bietet sich das Verwerten (engl. exploitation) schon gelerner \hat{Q} -Werte an. Der Agent wählt hier immer die Aktion mit dem höchsten \hat{Q} -Wert. Dies führt zu einer relativ schnellen Konvergenz bestimmter Trajektorien. Andere Wege hingegen bleiben bis zum Schluss unbesucht. Im Extremfall erhält man dann nicht optimale Strategien. In Satz 10.1 wird daher gefordert, dass jedes Zustands-Aktions-Paar unendlich oft besucht wird. Es empfiehlt sich eine Kombination aus Erkunden und Verwerten mit einem hohen Erkundungsanteil am Anfang, der dann im Laufe der Zeit immer weiter reduziert wird.

Auch die Wahl des Startzustandes beeinflusst die Geschwindigkeit des Lernens. In den ersten drei Bildern von Abb. 10.12 erkennt man gut, dass sich bei den ersten Iterationen nur die Q -Werte in unmittelbarer Umgebung von Zustands-Aktions-Paaren mit direkter Belohnung ändern. Ein Start in großer Entfernung von derartigen Punkten führt also zu viel unnötiger Arbeit. Es bietet sich daher an, entsprechendes Vorwissen über Zustands-Aktions-Paare mit direkter Belohnung umzusetzen in Startzustände nahe bei diesen Punkten. Im Laufe des Lernens können dann immer weiter entfernte Startzustände gewählt werden.

10.8 Approximation, Generalisierung und Konvergenz

So wie das Q-Lernen bisher beschrieben wurde, wird explizit eine Tabelle mit allen Q -Werten gespeichert. Dies ist nur möglich, wenn auf einem endlichen Zustandsraum mit endlich vielen Aktionen gearbeitet wird. Ist jedoch der Zustandsraum unendlich, zum Beispiel im Fall von stetigen Variablen, so ist es weder möglich, alle Q -Werte zu speichern, noch ist es möglich, beim Lernen alle Zustands-Aktionspaare zu besuchen.

Trotzdem gibt es eine einfache Möglichkeit, Q-Lernen und Wert-Iteration auf stetige Variablen anzuwenden. Die $Q(s, a)$ -Tabelle wird ersetzt durch ein neuronales Netz, zum Beispiel ein Backpropagation-Netz mit den Input-Variablen s, a und dem Q -Wert als Ziel-Output. Bei jeder Aktualisierung eines Q -Wertes wird dem neuronalen Netz ein Trainingsbeispiel mit (s, a) als Input und $Q(s, a)$ als Ziel-Output präsentiert. Am Ende hat man eine endliche Repräsentation der Funktion $Q(s, a)$. Da man immer nur endlich viele Trainingsbeispiele hat, die Funktion $Q(s, a)$ aber auf unendlich vielen möglichen Eingaben definiert ist, erhält man bei passend gewählter Netzwerkgröße (siehe Kap. 9) damit automatisch eine Generalisierung. Statt eines neuronalen Netzes kann man auch ein anderes überwachtes Lernverfahren, beziehungsweise einen Funktionsapproximator wie zum Beispiel eine Support-Vektor-Maschine wählen.

Allerdings kann der Schritt von endlich vielen Trainingsbeispielen auf eine stetige Funktion unter Umständen teuer werden. Es kann sein, dass das Q-Lernen mit Funktionsappro-

ximation nicht mehr konvergiert, denn Satz 10.1 gilt nur, wenn jedes Zustands-Aktionspaar unendlich oft besucht wird.

Probleme mit der Konvergenz können aber auch im Fall von endlich vielen Zustands-Aktionspaaren auftreten, wenn Q-Lernen auf einen POMDP angewendet wird. Q-Lernen ist – in den beiden beschriebenen Varianten – für deterministische und nichtdeterministische Markov-Prozesse (MDP) anwendbar. Bei einem POMDP kann es vorkommen, dass der Agent zum Beispiel aufgrund von fehlerhaften Sensoren viele verschiedene Zustände als einen erkennt. Oft werden auch ganz bewusst viele Zustände in der realen Welt auf eine so genannte **Beobachtung** (engl. observation) abgebildet. Der entstehende Beobachtungsraum ist dann viel kleiner als der Zustandsraum, wodurch das Lernen schneller wird und Überanpassung (engl. overfitting, siehe Abschn. 8.4.7) vermieden werden kann.

Der Agent kann durch das Zusammenfassen mehrerer Zustände aber nicht mehr zwischen den tatsächlichen Zuständen unterscheiden, und eine Aktion kann ihn, je nachdem, in welchem wirklichen Zustand er ist, in viele verschiedene Nachfolgezustände führen, was zu Konvergenzproblemen bei der Wert-Iteration oder beim Q-Lernen führen kann. In der Literatur (z. B. in [SB98]) werden verschiedene Lösungsansätze vorgeschlagen.

Vielversprechend sind auch die sogenannten Policy Improvement-Methoden und die daraus entstandenen Policy-Gradient-Methoden, bei denen iterativ nicht Q-Werte für Zustände oder Aktionen verändert werden, sondern direkt die Strategie. Bei diesen Verfahren wird im Raum aller Strategien eine Strategie mit maximalem Erwartungswert der Summe aller direkten Belohnungen über alle Aktionen gesucht. Eine Möglichkeit, dies zu erreichen, ist das Folgen des Gradienten der kumulativen direkten Belohnung bis zu einem Maximum. Die so gefundene Strategie maximiert dann offenbar die kumulative direkte Belohnung. In [PS08] wird gezeigt, dass diese Verfahren das Lernen in Anwendungen mit großen Zustandsräumen, wie sie zum Beispiel bei humanoiden Robotern vorkommen, stark beschleunigen können.

10.9 Anwendungen

Die praktische Anwendbarkeit des Lernens durch Verstärkung wurde mittlerweile schon vielfach bewiesen. Aus der großen Zahl von Beispielen hierfür stellen wir eine kleine Auswahl kurz vor.

Das TD-Lernen zusammen mit einem Backpropagation-Netz mit 40 bis 80 verdeckten Neuronen wurde sehr erfolgreich angewendet in TD-Gammon, einem Programm zum Spielen von Backgammon [Tes95]. Die einzige direkte Belohnung für das Programm ist das Ergebnis am Ende eines Spiels. Eine optimierte Version des Programms mit einer 2-Züge-Vorausschau wurde in 1,5 Millionen Spielen gegen sich selbst trainiert. Es besiegte damit Weltklassespieler und spielt so gut wie die drei besten menschlichen Spieler.

Viele Anwendungen gibt es in der Robotik. Zum Beispiel in der RoboCup Soccer Simulation League arbeiten die besten Roboter-Fußball-Teams heute erfolgreich mit Lernen

durch Verstärkung [SSK05, Robb]. Auch das für uns Menschen relativ leichte Balancieren eines Stabes wurde schon vielfach erfolgreich mit Lernen durch Verstärkung gelöst.

Eine beeindruckende Demonstration der Lernfähigkeit von Robotern gab Russ Tedrake auf der IROS 2008 durch seinen Vortrag über ein Modellflugzeug, das lernt, exakt auf einem Punkt zu landen, genau wie ein Vogel, der auf einem Ast landet [Ted08]. Da bei solch einem hochdynamischen Landeanflug die Strömung sehr turbulent wird, ist die zugehörige Differentialgleichung, die Navier-Stokes-Gleichung, nicht lösbar. Auf dem klassischen mathematischen Weg kann das Landen daher nicht gesteuert werden. Tedrake's Kommentar dazu:

Birds don't solve Navier-Stokes!

Vögel können offenbar auch ohne Navier-Stokes-Gleichung das Fliegen und Landen lernen. Tedrake zeigte, dass dies nun auch mit Flugzeugen möglich ist.

Auch ist es heute schon möglich, durch Q-Lernen mit Funktionsapproximation die Steuerung eines realen Autos in nur 20 Minuten zu lernen [RMD07]. Dieses Beispiel zeigt, dass reale industrielle Anwendungen, bei denen wenige Messgrößen auf eine Steuergröße abgebildet werden müssen, sehr gut und auch schnell gelernt werden können.

Schwierigkeiten gibt es noch beim Lernen auf realen Robotern in hochdimensionalen Zustands-Aktionsräumen, denn im Vergleich zu einer Simulation erfolgt das Feedback der Umwelt bei realen Robotern relativ langsam. Aus Zeitgründen sind die vielen Millionen benötigten Trainingszyklen daher nicht realisierbar. Hier werden neben schnelleren Lernalgorithmen auch Methoden benötigt, die zumindest Teile des Lernens offline, das heißt ohne Feedback von der Umgebung, erlauben.

10.10 Fluch der Dimensionen

Trotz der Erfolge in den letzten Jahren bleibt das Lernen durch Verstärkung ein sehr aktives Forschungsgebiet der KI, nicht zuletzt deshalb, weil auch die besten heute bekannten Lernalgorithmen bei hochdimensionalen Zustands- und Aktionsräumen wegen ihrer gigantischen Rechenzeiten immer noch nicht praktisch anwendbar sind. Dieses Problem wird als „curse of dimensionality“ (Fluch der Dimensionalität) bezeichnet.

Auf der Suche nach Lösungen für dieses Problem beobachten die Wissenschaftler Tiere oder Menschen beim Lernen. Dabei fällt auf, dass das Lernen in der Natur auf vielen Abstraktionsebenen abläuft. Ein Baby lernt zuerst auf der untersten Ebene einfache motorische oder auch sprachliche Fähigkeiten. Wenn diese gut gelernt sind, werden sie gespeichert und können später jederzeit abgerufen und angewendet werden. In die Sprache der Informatik übersetzt heißt das, dass jede gelernte Fähigkeit in ein Modul gekapselt wird und dann auf höherer Ebene eine Aktion darstellt. Durch die Verwendung komplexer Aktionen wird der Aktionsraum stark verkleinert und damit das Lernen beschleunigt. In ähnlicher Weise können auch die Zustände abstrahiert und damit der Zustandsraum redu-

ziert werden. Dieses Lernen auf mehreren Ebenen wird als hierarchisches Lernen [BM03] bezeichnet.

Ein anderer Ansatz zur Modularisierung des Lernens ist das verteilte Lernen oder auch Multi-Agenten-Lernen [PL05]. Beim Lernen motorischer Fähigkeiten eines humanoiden Roboters müssen bis zu 50 verschiedene Motoren gleichzeitig angesteuert werden, was zu einem 50-dimensionalen Zustandsraum und auch einem 50-dimensionalen Aktionsraum führt. Um diese gigantische Komplexität zu verringern, wird die zentrale Steuerung ersetzt durch eine verteilte. Zum Beispiel könnte jeder einzelne Motor eine eigene Steuerung bekommen, die diesen direkt steuert, wenn möglich unabhängig von den anderen Motoren. In der Natur findet man derartige Steuerungen bei Insekten. Zum Beispiel werden die vielen Beine eines Tausendfüßlers nicht von einem zentralen Gehirn gesteuert, sondern jedes Beinpaar besitzt ein eigenes kleines „Gehirn“.

Ähnlich wie bei der uninformierten kombinatorischen Suche stellt sich beim Lernen durch Verstärkung die Aufgabe, aus der riesigen Anzahl von Strategien eine Beste zu finden. Die Aufgabe des Lernens wird wesentlich einfacher, wenn der Agent schon bevor er mit dem Lernen beginnt, eine mehr oder weniger gute Strategie besitzt. Dann können die hochdimensionalen Lernaufgaben eher gelöst werden. Wie findet man aber solch eine initiale Strategie? Hier gibt es zwei prinzipielle Möglichkeiten.

Die erste Möglichkeit ist die klassische Programmierung. Der Programmierer gibt dem Agenten per Programm eine Strategie vor, die er für gut hält. Dann wird umgeschaltet, zum Beispiel auf das Q-Lernen. Der Agent wählt zumindest am Anfang des Lernens seine Aktionen entsprechend der programmierten Strategie und wird so in „interessante“ Bereiche des Zustands-Aktionsraumes geführt.

Wird die klassische Programmierung zu komplex, kann man das Trainieren des Roboters oder Agenten beginnen, indem ein Mensch die richtigen Aktionen vorgibt. Im einfachsten Fall erfolgt dies durch eine manuelle Fernsteuerung des Roboters. Dieser speichert dann für jeden Zustand die vorgegebene Aktion und generalisiert nun mittels eines überwachten Lernverfahrens wie zum Beispiel Backpropagation oder Entscheidungsbaumlernern. Dieses sogenannte Lernen durch Demonstration [BCDS08, SE10] liefert also auch eine initiale Strategie für das dann folgende Lernen durch Verstärkung.

10.11 Zusammenfassung und Ausblick

Wir verfügen heute über gut funktionierende und etablierte Lernalgorithmen um unsere Maschinen zu trainieren. Die Aufgabe für den menschlichen Trainer oder Entwickler ist aber bei komplexen Anwendungen immer noch anspruchsvoll. Es gibt nämlich viele Möglichkeiten, wie er das Training eines Roboters gestalten kann und er wird ohne Experimentieren nicht erfolgreich sein. Dieses Experimentieren kann in der Praxis sehr mühsam sein, denn jedes neue Lernprojekt muss entworfen und programmiert werden. Hier sind Werkzeuge gefragt, die dem Trainer neben den verschiedenen Lernalgorithmen auch die erwähnten Kombinationen mit der klassischen Programmierung und dem Lernen durch

Demonstration anbieten. Ein erstes derartiges Werkzeug ist die Teaching-Box [ESCT09], welche neben einer umfangreichen Programmbibliothek auch eine Benutzerschnittstelle zur Konfiguration von Lernprojekten und zur Kommunikation des menschlichen Lehrers mit dem Roboter bietet. Zum Beispiel kann der menschliche Lehrer dem Roboter zusätzlich zum Feedback aus der Umgebung noch weiteres Feedback über die Tastatur oder eine Sprachschnittstelle geben.

Das Lernen durch Verstärkung ist ein faszinierendes und aktives Forschungsgebiet und es wird in der Zukunft immer mehr in Anwendungen benutzt werden. Immer mehr Robotersteuerungen, aber auch andere Programme werden durch das Feedback der Umgebung lernen. Es existieren heute eine Vielzahl an Variationen der vorgestellten Algorithmen und auch ganz andere Verfahren. Noch nicht gelöst ist das Problem der Skalierung. Für kleine Aktions- und Zustandsräume mit wenigen Freiheitsgraden lassen sich beeindruckende Ergebnisse erzielen. Steigt die Zahl der Freiheitsgrade im Zustandsraum zum Beispiel auf 18 für einen einfachen humanoiden Roboter, so wird das Lernen sehr aufwändig.

Zur weiteren grundlegenden Lektüre empfehlen wir die gute und kompakte Einführung in das Lernen durch Verstärkung in dem Buch von Tom Mitchell [Mit97]. Sehr ausführlich und umfassend ist das Standardwerk von Sutton und Barto [SB98] sowie der Übersichtsartikel von Kaelbling, Littman und Moore [KLM96].

10.12 Übungen

Aufgabe 10.1

- Berechnen Sie die Zahl unterschiedlicher Strategien bei n Zuständen und n Aktionen.
Es sind also Übergänge von jedem Zustand in jeden Zustand möglich.
- Wie ändert sich die Zahl der Strategien in Teilaufgabe a, wenn leere Aktionen, das heißt Übergänge von einem Zustand auf sich selbst, nicht erlaubt sind.
- Geben Sie mit Hilfe von Pfeildiagrammen wie in Abb. 10.3 alle Strategien für 2 Zustände an.
- Geben Sie mit Hilfe von Pfeildiagrammen wie in Abb. 10.3 alle Strategien ohne leere Aktionen für 3 Zustände an.

Aufgabe 10.2

Wenden Sie die Wert-Iteration manuell auf das Beispiel 10.1 mit $n_x = n_y = 2$ an.

Aufgabe 10.3

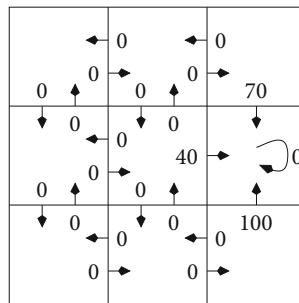
Mit Hilfe des Simulators für die Wert-Iteration sollen verschiedene Experimente durchgeführt werden.

- Installieren Sie den Simulator für die Wert-Iteration aus [Tok06].
- Reproduzieren Sie die Ergebnisse aus Aufgabe 10.2, indem Sie mit dem Feedback-Editor zuerst das Feedback einstellen und dann die Wert-Iteration ausführen.
- Modellieren Sie unterschiedlich glatte Untergründe und beobachten Sie, wie sich die Strategie ändert.

- d) Vergrößern Sie bei ähnlicher Feedbackmatrix den Zustandsraum schrittweise bis auf etwa 100×100 und passen Sie jeweils den Abschwächungsfaktor γ so an, dass sich eine sinnvolle Strategie ergibt.

Aufgabe 10.4 * Zeigen Sie, dass für die Beispielrechnung in Abb. 10.8 der genaue Wert $V^*(3, 3) = 1,9/(1 - 0,9^6) \approx 4,05499$ ist.

Aufgabe 10.5 Führen Sie das Q-Lernen auf dem untenstehenden 3×3 -Gitter durch. Der Zustand in der Mitte rechts ist ein absorbierender Endzustand.



Aufgabe 10.6 Gegeben sei ein Roboterarm mit n Gelenken (Dimensionen) und pro Gelenk ℓ diskreten Zuständen. Es seien Aktionen von jedem Zustand in jeden Zustand (d. h. wenn der Roboter nichts tut, wird das auch als (leere) Aktion gewertet) möglich.

- Geben Sie je eine Formel für die Zahl der Zustände, die Zahl der Aktionen in jedem Zustand und für die Zahl der Strategien des Roboters an.
- Erstellen Sie eine Tabelle mit der Zahl der Strategien für $n = 1, 2, 3, 4, 8$ und $\ell = 1, 2, 3, 4, 10$ an.
- Um die Zahl der möglichen Strategien zu reduzieren nehmen Sie nun an, dass die Zahl der möglichen Aktionen pro Gelenk immer gleich 2 ist und dass der Roboter zu einem Zeitpunkt nur ein Gelenk bewegen kann. Geben Sie eine neue Formel für die Zahl der Strategien an und erstellen Sie zugehörige Tabelle neu.
- Begründen Sie auf den berechneten Resultaten, dass man einen Agenten, der autonom und adaptiv mit $n = 8$ und $\ell = 10$ agiert, durchaus als intelligent bezeichnen kann.

11.1 Einführung

Aufgabe 1.3 Eine Maschine kann sehr wohl intelligent sein, auch wenn ihr Verhalten sich stark von dem eines Menschen unterscheidet.

Aufgabe 1.4 Wenn ein Problem NP-vollständig ist oder als „schwierig“ bezeichnet wird, so heißt das, dass es Instanzen des Problems gibt, die in akzeptabler Zeit nicht lösbar sind. Dies ist der so genannte *worst case*. Wir müssen wohl bei vielen Anwendungen damit leben, dass im worst-case eine effiziente Lösung nicht möglich ist. Das heißt, es wird auch in Zukunft praktisch relevante Probleme geben, die in bestimmten Spezialfällen unlösbar sind.

Die KI wird also weder eine Weltformel finden, noch eine Supermaschine bauen, mit der alle Probleme lösbar werden. Sie stellt sich daher vielmehr die Aufgabe, Systeme zu bauen, die mit hoher Wahrscheinlichkeit eine Lösung finden, bzw. mit hoher Wahrscheinlichkeit fast optimale Lösungen finden. Wir Menschen leben im Alltag fast immer mit suboptimalen Lösungen sehr gut. Der Grund ist ganz einfach der viel zu hohe Aufwand für das Finden der optimalen Lösung. Um zum Beispiel in einer fremden Stadt nach Stadtplan einen Weg von A nach B zu gehen, brauche ich 7 Minuten. Der kürzeste Weg hätte mich nur 6 Minuten gekostet. Das Finden des kürzesten Weges jedoch hätte mich vielleicht eine Stunde gekostet. Der Beweis für die Optimalität des Weges wäre vielleicht noch aufwändiger gewesen.

Aufgabe 1.5

- Die Ausgabe hängt nicht nur von der Eingabe, sondern auch noch vom Inhalt des Gedächtnisses ab. Für eine Eingabe x kann die Ausgabe je nach Gedächtnisinhalt also y_1 oder y_2 lauten. Sie ist also nicht eindeutig und der Agent daher keine Funktion.
- Betrachtet man den Inhalt des Gedächtnisses als weitere Eingabe, dann ist die Ausgabe eindeutig (weil der Agent deterministisch ist) und der Agent ist funktional.

Aufgabe 1.6

- a) Geschwindigkeit

$$v_x(t) = \frac{\partial x}{\partial t} = \lim_{\Delta t \rightarrow 0} \frac{x(t) - x(t - \Delta t)}{\Delta t} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}.$$

v_y berechnet sich analog.

- b) Beschleunigung

$$\begin{aligned} a_x(t) &= \frac{\partial^2 x}{\partial t^2} = \frac{\partial}{\partial t} v_x(t) = \lim_{\Delta t \rightarrow 0} \frac{v_x(t) - v_x(t - \Delta t)}{\Delta t} \\ &= \lim_{\Delta t \rightarrow 0} \left[\frac{x(t) - x(t - \Delta t)}{(\Delta t)^2} - \frac{x(t - \Delta t) - x(t - 2\Delta t)}{(\Delta t)^2} \right] \\ &= \frac{x(t) - 2x(t - \Delta t) + x(t - 2\Delta t)}{(\Delta t)^2} \end{aligned}$$

a_y berechnet sich analog. Man benötigt also die Position zu den drei Zeiten $t - 2\Delta t$, $t - \Delta t$, t .

Aufgabe 1.7

- a) Kosten für Agent 1 = $11 \cdot 100 \text{ Cent} + 1 \cdot 1 \text{ Cent} = 1101 \text{ Cent}$.

Kosten für Agent 2 = $0 \cdot 100 \text{ Cent} + 38 \cdot 1 \text{ Cent} = 38 \text{ Cent}$.

Agent 2 spart also $1101 \text{ Cent} - 38 \text{ Cent} = 1063 \text{ Cent}$ an Kosten.

- b) Nutzen für Agent 1 = $189 \cdot 100 \text{ Cent} + 799 \cdot 1 \text{ Cent} = 19.699 \text{ Cent}$.

Nutzen für Agent 2 = $200 \cdot 100 \text{ Cent} + 762 \cdot 1 \text{ Cent} = 20.762 \text{ Cent}$.

Agent 2 hat also $20.762 \text{ Cent} - 19.699 \text{ Cent} = 1063 \text{ Cent}$ höheren Nutzen. Setzt man für Fehlentscheidungen den entgangenen Nutzen an, so kommt ein nutzenorientierter Agent also zu gleichen Entscheidungen wie ein kostenorientierter.

11.2 Aussagenlogik

Aufgabe 2.1 Mit der Signatur $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$ und der Grammatikvariablen $\langle \text{Formel} \rangle$ lässt sich die Syntax der Aussagenlogik wie folgt definieren:

$$\begin{aligned} \langle \text{Formel} \rangle ::= & \sigma_1 \mid \sigma_2 \mid \dots \mid \sigma_n \mid w \mid f \\ & \mid \neg \langle \text{Formel} \rangle \mid (\langle \text{Formel} \rangle) \mid \langle \text{Formel} \rangle \wedge \langle \text{Formel} \rangle \\ & \mid \langle \text{Formel} \rangle \vee \langle \text{Formel} \rangle \mid \langle \text{Formel} \rangle \Rightarrow \langle \text{Formel} \rangle \\ & \mid \langle \text{Formel} \rangle \Leftrightarrow \langle \text{Formel} \rangle \end{aligned}$$

Aufgabe 2.2 Nachweis mit Wahrheitstafelmethode

Aufgabe 2.3 a) $(\neg A \vee B) \wedge (\neg B \vee A)$ b) $(\neg A \vee B) \wedge (\neg B \vee A)$ c) w

Aufgabe 2.4 a) erfüllbar b) wahr c) unerfüllbar

Aufgabe 2.6

- a) In Aufgabe 2.3 c wurde schon gezeigt, dass $A \wedge (A \Rightarrow B) \Rightarrow B$ eine Tautologie ist.
Das Deduktionstheorem stellt dann die Korrektheit der Inferenzregel sicher.
- b) Man zeigt per Wahrheitstafelmethode, dass $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$ eine Tautologie ist.

Aufgabe 2.7 Anwendung der Resolutionsregel auf die Klauseln $(f \vee B)$ und $(\neg B \vee f)$ ergibt die Resolvente $(f \vee f) \equiv (f)$. Nun wenden wir die Resolutionsregel auf die Klauseln B und $\neg B$ an und erhalten die leere Klausel als Resolvente. Da $(f \vee B) \equiv B$ und $(\neg B \vee f) \equiv \neg B$ ist $(f) \equiv ()$. Für die Praxis wichtig ist, dass, wann immer die leere Klausel abgeleitet wird, ein Widerspruch die Ursache ist.

Aufgabe 2.8 Enthält WB einen Widerspruch, so lassen sich zwei Klauseln A und $\neg A$ und daraus die leere Klausel ableiten. Der Widerspruch in WB ist natürlich in $WB \wedge \neg Q$ immer noch enthalten. Also lässt sich auch hier die leere Klausel ableiten.

Aufgabe 2.9 a) $(A \vee B) \wedge (\neg A \vee \neg B)$ b) $(A \vee B) \wedge (B \vee C) \wedge (A \vee C)$

Aufgabe 2.10 Formalisierung: Komplize: K , Wagen: W , Schlüssel: S

$$WB \equiv (K \Rightarrow W) \wedge [(\neg K \wedge \neg S) \vee (K \wedge S)] \wedge S$$

Transformation in KNF: $(\neg K \wedge \neg S) \vee (K \wedge S) \equiv (\neg S \vee K) \wedge (\neg K \vee S)$

Versuche W zu beweisen und füge daher $\neg W$ zur Klauselmenge dazu. Als KNF-Darstellung ergibt sich:

$$(\neg K \vee W)_1 \wedge (\neg S \vee K)_2 \wedge (\neg K \vee S)_3 \wedge (S)_4 \wedge (\neg W)_5.$$

Resolutionsbeweis: Res(2, 4): $(K)_6$
 Res(1, 6): $(W)_7$
 Res(7, 5): $()_8$

Damit ist W bewiesen.

Aufgabe 2.11

- a) $WB \equiv (A \vee B) \wedge (\neg B \vee C), Q \equiv (A \vee C)$
 $WB \wedge \neg Q \equiv (A \vee B)_1 \wedge (\neg B \vee C)_2 \wedge (\neg A)_3 \wedge (\neg C)_4$

Resolutionsbeweis: Res(1, 3): $(B)_5$
 Res(2, 4): $(\neg B)_6$
 Res(5, 6): $()$

b) $\neg(\neg B \wedge (B \vee \neg A)) \Rightarrow \neg A \equiv (\neg B)_1 \wedge (B \vee \neg A)_2 \wedge (A)_3$

Resolutionsbeweis: Res(1, 2): $(\neg A)_4$
 Res(3, 4): $()$

Aufgabe 2.12 Unter Verwendung der Äquivalenzen aus Satz 2.1 kann man sofort die Behauptungen zeigen.

Aufgabe 2.13

- Res(8, 9): $(C \wedge F \wedge E \Rightarrow f)_{10}$
 Res(3, 10): $(F \wedge E \Rightarrow f)_{11}$
 Res(6, 11): $(A \wedge B \wedge C \wedge E \Rightarrow f)_{12}$
 Res(1, 12): $(B \wedge C \wedge E \Rightarrow f)_{13}$
 Res(2, 13): $(C \wedge E \Rightarrow f)_{14}$
 Res(3, 14): $(E \Rightarrow f)_{15}$
 Res(5, 15): $()$

11.3 Prädikatenlogik

Aufgabe 3.1

- a) $\forall x \text{ männlich}(x) \Leftrightarrow \neg \text{weiblich}(x)$
 b) $\forall x \forall y \exists z \text{ vater}(x, y) \Leftrightarrow \text{männlich}(x) \wedge \text{kind}(y, x, z)$
 c) $\forall x \forall y \text{ geschwister}(x, y) \Leftrightarrow [(\exists z \text{ vater}(z, x) \wedge \text{vater}(z, y)) \vee (\exists z \text{ mutter}(z, x) \wedge \text{mutter}(z, y))]$
 d) $\forall x \forall y \forall z \text{ eltern}(x, y, z) \Leftrightarrow \text{vater}(x, z) \wedge \text{mutter}(y, z)$.
 e) $\forall x \forall y \text{ onkel}(x, y) \Leftrightarrow \exists z \exists u \text{ kind}(y, z, u) \wedge \text{geschwister}(z, x) \wedge \text{männlich}(x)$.
 f) $\forall x \forall y \text{ vorfahr}(x, y) \Leftrightarrow \exists z \text{ kind}(y, x, z) \vee \exists u \exists v \text{ kind}(u, x, v) \wedge \text{vorfahr}(u, y))$.

Aufgabe 3.2

- a) $\forall x \exists y \exists z \text{ vater}(y, x) \wedge \text{mutter}(z, x)$
 b) $\exists x \exists y \exists z \text{ kind}(y, x, z)$
 c) $\forall x \text{ vogel}(x) \Rightarrow \text{fliegt}(x)$
 d) $\exists x \exists y \exists z \text{ tier}(x) \wedge \text{tier}(y) \wedge \text{frisst}(x, y) \wedge \text{frisst}(y, z) \wedge \text{korn}(z)$
 e) $\forall x \text{ tier}(x) \Rightarrow (\exists y (\text{frisst}(x, y) \wedge (\text{pflanze}(y) \vee (\text{tier}(y) \wedge \exists z \text{ pflanze}(z) \wedge \text{frisst}(y, z) \wedge \text{viel_kleiner}(y, x))))$

Aufgabe 3.3 $\forall x \forall y \exists z x = \text{vater}(y) \Leftrightarrow \text{männlich}(x) \wedge \text{kind}(y, x, z)$

Aufgabe 3.4 $\forall x \forall y x < y \vee y < x \vee x = y \wedge \neg(x < y \wedge y < x) \wedge \neg(x < y \wedge x = y) \wedge \neg(y < x \wedge x = y)$,
 $\forall x \forall y x < y \Rightarrow \neg y < x$,
 $\forall x \forall y \forall z x < y \wedge y < z \Rightarrow x < z$

Aufgabe 3.5

- a) MGU: $x/f(z), u/f(y)$, Term: $p(f(z), f(y))$
 b) nicht unifizierbar
 c) MGU: $x/\cos y, z/4 - 7 \cdot \cos y$, Term: $\cos y = 4 - 7 \cdot \cos y$
 d) nicht unifizierbar
 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) Sei die unerfüllbare Formel $p(x) \wedge \neg p(x) \wedge r(x)$ gegeben. Wählen wir nun die Klausel $r(x)$ als SOS, so kann kein Widerspruch abgeleitet werden.
 b) Ist schon das SOS unerfüllbar, so kann daraus ein Widerspruch abgeleitet werden. Falls nicht, so sind Resolutionsschritte zwischen Klauseln aus SOS und $(WB \wedge \neg Q) \setminus SOS$ nötig.
 c) Gibt es zu einem Literal L in einer Klausel K kein komplementäres, so wird in jeder Klausel, die mittels Resolution aus der Klausel K abgeleitet wird, das Literal L immer stehen bleiben. Also kann weder aus K noch aus deren Resolventen und zukünftigen Resolventen die leere Klausel abgeleitet werden.

Aufgabe 3.8 $\neg Q \wedge WB \equiv (e = n)_1 \wedge (n \cdot x = n)_2 \wedge (e \cdot x = x)_3 \wedge (\neg a = b)_4$

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): $(\neg e = e)_8$

()

Hier steht „Dem“ für Demodulation. Klausel Nr. 6 wurde hergeleitet durch Anwendung von Transitivität und Symmetrie der Gleichheit aus den Klauseln 3 und 5.

Aufgabe 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) .
 $raisiert (barb, X) ; rasiert (X, X) < - .$

c) $e = n.$
 $\leftarrow a = b.$
 $m(m(X, Y), Z) = m(X, m(Y, Z)).$
 $m(e, X) = X.$
 $m(n, X) = n.$

11.4 Grenzen der Logik

Aufgabe 4.1

- a) Korrekt ist: *Man nehme einen vollständigen Beweiskalkül für PL1. Für jede wahre Formel findet man damit in endlicher Zeit einen Beweis. Für alle unerfüllbaren Formeln ϕ gehe ich wie folgt vor: Ich wende den Kalkül auf $\neg\phi$ an und zeige, dass $\neg\phi$ wahr ist. Also ist ϕ falsch. Damit kann man jede wahre Formel aus PL1 beweisen und jede falsche Formel widerlegen.* Für erfüllbare Formeln taugt dieses Verfahren leider nicht.
- b) Sei $\phi \in PL1$ wahr oder unerfüllbar. Wendet man einen Beweiser parallel auf ϕ und auf $\neg\phi$ an, so wird er nach endlicher Zeit eine von beiden beweisen. Also ist diese Teilmenge von PL1 entscheidbar.

Aufgabe 4.2

- a) Er rasiert sich genau dann, wenn er sich nicht rasiert. (Widerspruch)
- b) Die Menge aller Mengen, die sich nicht selbst enthalten. Sie enthält sich genau dann, wenn sie sich nicht selbst enthält.

11.5 Prolog

Aufgabe 5.1 Prolog meldet einen Stacküberlauf. Der Grund ist die Tiefensuche von Prolog, die immer das erste unifizierbare Prädikat in der Eingabedatei auswählt. Bei rekursiven Prädikaten wie zum Beispiel der Gleichheit entsteht daraus eine nicht terminierende Rekursion.

Aufgabe 5.2

```
write_path( [H1,H2|T] ) :- write_path([H2|T]), write_move(H2,H1).
write_path( [_X] ) .
```

```
write_move( zust(X,W,Z,K), zust(Y,W,Z,K) ) :-
    write('Bauer von '), write(X), write(' nach '), write(Y), nl.
```

```
write_move( zust(X,X,Z,K), zust(Y,Y,Z,K) ) :-
    write('Bauer und Wolf von '), write(X), write(' nach '), write(Y),nl.
```

```
write_move( zust(X,W,X,K), zust(Y,W,Y,K) ) :-
    write('Bauer und Ziege von '), write(X), write(' nach '), write(Y), nl.
```

```
write_move( zust(X,W,Z,X), zust(Y,W,Z,Y) ) :-
    write('Bauer und Kohl von '), write(X), write(' nach '), write(Y), nl.
```

Aufgabe 5.3

- a) Sie wird benötigt, um die gefundene Lösung auszugeben. Die Unifikation im Fakt `plan(Ziel, Ziel, Pfad, Pfad)`. sorgt dafür.
- b) Die Bedingungen zum Eintritt in die Klauseln des Prädikates `sicher` überlappen sich. Das durch den `fail` verursachte Backtracking führt zum Ausführen der zweiten oder dritten Alternative von `sicher`, wobei offenbar die gleiche Lösung nochmal gefunden wird. Ein Cut am Ende der ersten beiden `sicher`-Klauseln löst das Problem. Alternativ können auch alle sichereren Zustände explizit angegeben werden, wie z. B. `sicher(zust(links, links, links, rechts))`.

Aufgabe 5.5

```
?- eins(10).
eins(0) :- write(1).
eins(N) :- N1 is N-1, eins(N1), eins(N1).
```

Aufgabe 5.6

- b) Mit $n_1 = |L1|, n_2 = |L2|$ gilt $T_{\text{append}}(n_1, n_2) = \Theta(n_1)$.
- c) Mit $n = |L|$ gilt $T_{\text{nrev}}(n) = \Theta(n^2)$.
- d) Mit $n = |L|$ gilt $T_{\text{accrev}}(n) = \Theta(n)$.

Aufgabe 5.7 Für die Bäume mit Symbolen an den inneren Knoten kann man zum Beispiel die Terme `a(b, c)` und `a(b(e, f, g), c(h), d)` verwenden. Bäume ohne Symbole: `baum(b, c)`, bzw. `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 is N-1, fib(N1,R1), N2 is N-2, fib(N2,R2),
R is R1 + R2.
```
- b)  $T(n) = \Theta(2^n)$ .
- c) `: - dynamic fib/2. fib(0,1). fib(1,1).`
- ```
fib(N,R) :- N1 is N-1, fib(N1,R1), N2 is N-2, fib(N2,R2),
R is R1 + R2, asserta(fib(N,R)).
```
- d) Wenn die Fakten `fib(0,1)` bis `fib(k, fib(k))` schon berechnet wurden, gilt für den Aufruf `fib(n,X)`

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n \leq k \\ \Theta(n-k) & \text{falls } n > k \end{cases}$$

- e) Weil nach den ersten n Aufrufen des `fib`-Prädikates alle weiteren sofort auf Fakten zugreifen.

Aufgabe 5.9

a) Die Lösung wird hier nicht verraten.

b) start :-

```

fd_domain([Brite, Schwede, Daene, Norweger, Deutscher],1,5),
fd_all_different([Brite, Schwede, Daene, Norweger, Deutscher]),
fd_domain([Tee, Kaffee, Wasser, Bier, Milch],1,5),
fd_all_different([Tee, Kaffee, Wasser, Bier, Milch]),
fd_domain([Rot, Weiss, Gruen, Gelb, Blau],1,5),
fd_all_different([Rot, Weiss, Gruen, 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, Daene, Norweger, Deutscher]),
Brite #= Rot,                      % Der Brite lebt im roten Haus
Schwede #= Hund,                   % Der Schwede hält einen Hund
Daene #= Tee,                      % Der Däne trinkt gerne Tee
Gruen #= Weiss - 1,                % Das grüne Haus steht links vom weißen Haus
Gruen #= Kaffee,                   % Der Besitzer vom grünen Haus trinkt Kaffee
Pallmall #= Vogel,                 % Die Person, die Pall Mall raucht, hat e. 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,Katze)#= 1,          % Der Marlbororaucher wohnt neben der Katze
dist(Pferd,Dunhill) #= 1,           % Der Mann mit Pferd wohnt neben d. Dunhill-Ra.
Winfield #= Bier,                  % Der Winfieldraucher trinkt gerne Bier
dist(Norweger,Blau) #= 1,           % Der Norweger wohnt neben dem blauen Haus
Deutscher #= Rothmanns,            % Der Deutsche raucht Rothmanns
dist(Marlboro,Wasser)#=1,          % Der Marlborora. hat Nachbarn d. Wasser trinkt
write([Brite, Schwede, Daene, Norweger, Deutscher]), nl,
write([Hund, Vogel, Katze, Pferd, Fisch]), nl.

```

11.6 Suchen, Spielen und Probleme lösen

Aufgabe 6.1

a) Auf der letzten Ebene gibt es b^d Knoten. Alle Ebenen davor haben zusammen

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

Knoten, wenn b groß wird. Wegen $b^d/b^{d-1} = b$ gibt es auf der letzten Ebene also etwa b -mal so viele Knoten wie auf allen anderen Ebenen zusammen.

b) Bei nicht konstantem Verzweigungsfaktor gilt die Aussage des Satzes nicht mehr, wie folgendes Gegenbeispiel zeigt: Ein Baum, der bis zur vorletzten Ebene stark verzweigt, gefolgt von einer Ebene mit konstantem Verzweigungsfaktor 1, hat auf der letzten Ebene gleich viele Knoten wie auf der vorletzten Ebene.

Aufgabe 6.2

- a) In Abb. 6.3 wiederholt sich die Struktur des Baumes ab der zweiten Ebene mit ihren acht Knoten. Daher können wir den mittleren Verzweigungsfaktor b_m berechnen aus $b_m^2 = 8$ zu $b_m = \sqrt{8}$.
- b) Die Berechnung ist hier nicht ganz so einfach, denn der Wurzelknoten des Baumes verzweigt stärker als alle anderen gleichartigen. Man kann aber sagen, dass im Inneren des Baumes der Verzweigungsfaktor exakt um 1 kleiner ist als ohne Zyklenschluss. Daher gilt hier $b_m \approx \sqrt{8} - 1 \approx 1,8$.

Aufgabe 6.3

- a) Beim mittleren Verzweigungsfaktor wird die Zahl der Blattknoten fixiert. Beim effektiven hingegen die Zahl der Knoten im ganzen Baum.
- b) Weil die Zahl aller Knoten im Baum meist ein besseres Maß für die Rechenzeit zum Durchsuchen des Baumes darstellt als die Zahl der Blattknoten.
- c) Für großes b gilt nach (6.1) $n \approx \tilde{b}^{d+1}/\tilde{b} = \tilde{b}^d$, woraus $\tilde{b} = \sqrt[d]{n}$.

Aufgabe 6.4

- a) 3-Puzzle: $4! = 24$ Zustände, 8-Puzzle: $9! = 362.880$ Zustände, 15-Puzzle: $16! = 20.922.789.888.000$ Zustände.
- b) Nach 12mal im Uhrzeigersinn verschieben des leeren Feldes erreicht man wieder den Startzustand und erzeugt so einen zyklischen Unterraum mit 12 Zuständen.

Aufgabe 6.6

- a) Mathematica-Programm:

```

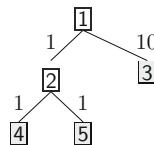
0 Breitensuche[Knoten_, Ziel_, Tiefe_] := Module[{i, NeueKnoten={}},
1   For[i=1, i<= Length[Knoten], i++,
2     NeueKnoten = Join[ NeueKnoten, Nachfolger[ Knoten[[i]] ] ];
3     If[MemberQ[Nachfolger[ Knoten[[i]] ], Ziel],
4       Return[{"Lösung gefunden", Tiefe+1}], 0
5     ]
6   ];
7   If[Length[NeueKnoten] > 0,
8     Breitensuche[NeueKnoten, Ziel, Tiefe+1],
9     Return[{"Fail", Tiefe}]
10    ]
11  ]

```

- b) Weil die Tiefe des Suchraums nicht beschränkt ist.

Aufgabe 6.7

- a) Bei konstanten Kosten der Höhe 1 sind die Kosten aller Pfade auf Tiefe d kleiner als die Kosten aller Pfade auf Tiefe $d + 1$. Da vor dem ersten Knoten auf Tiefe $d + 1$ alle Knoten auf Tiefe d getestet werden, wird eine Lösung der Länge d garantiert vor einer Lösung der Länge $d + 1$ gefunden.
- b) Bei untenstehendem Suchbaum werden zuerst Knoten Nummer 2 und der Lösungsknoten Nummer 3 mit Kosten in Höhe von 10 erzeugt. Die Suche terminiert. Die Knoten Nummer 4 und 5 mit Pfadkosten von jeweils 2 werden nicht erzeugt. Die optimale Lösung wird also nicht gefunden.

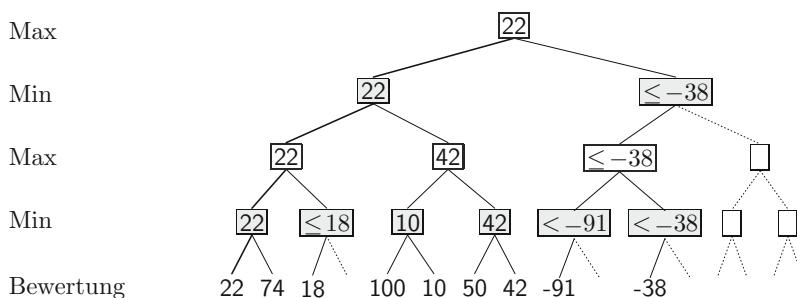


Aufgabe 6.11 Tiefensuche: Bewertung der neuen Knoten muss kleiner sein als die aller offenen Knoten. Breitensuche: Bewertung der neuen Knoten muss größer sein als die aller offenen Knoten.

Aufgabe 6.12 Genau wie eine zulässige Heuristik unterschätzt die Dame eventuell die Entfernung zum Ziel. Dies könnte dazu führen, dass die beiden – zwar unter großen Mühen – den schnellsten Weg zu den Blumen finden. Dies gilt allerdings nur dann, wenn die Dame die Entfernung immer unterschätzt.

Aufgabe 6.14

a) Max



11.7 Schließen mit Unsicherheit

Aufgabe 7.1

1. $P(\Omega) = \frac{|\Omega|}{|\Omega|} = 1$.
2. $P(\emptyset) = 1 - P(\Omega) = 0$.
3. Für paarweise unvereinbare Ereignisse A und B gilt $P(A \vee B) = \frac{|A \vee B|}{|\Omega|} = \frac{|A|+|B|}{|\Omega|} = P(A) + P(B)$.
4. Für zwei komplementäre Ereignisse A und $\neg A$ gilt $P(A) + P(\neg A) = P(A) + P(\Omega - A) = \frac{|A|}{|\Omega|} + \frac{|\Omega - A|}{|\Omega|} = \frac{|\Omega|}{|\Omega|} = 1$.
5. $P(A \vee B) = \frac{|A \vee B|}{|\Omega|} = \frac{|A|+|B|-|A \wedge B|}{|\Omega|} = P(A) + P(B) - P(A \wedge B)$.
6. Für $A \subseteq B$ gilt $P(A) = \frac{|A|}{|\Omega|} \leq \frac{|B|}{|\Omega|} = P(B)$.
7. Nach Definition 7.1 ist $A_1 \vee \dots \vee A_n = \Omega$ und $\sum_{i=1}^n P(A_i) = \sum_{i=1}^n \frac{|A_i|}{|\Omega|} = \frac{\sum_{i=1}^n |A_i|}{|\Omega|} = \frac{|A_1 \vee \dots \vee A_n|}{|\Omega|} = \frac{|\Omega|}{|\Omega|} = 1$.

Aufgabe 7.2

a) $P(y > 3 | Klasse = gut) = 7/9 \quad P(Klasse = gut) = 9/13 \quad P(y > 3) = 7/13$

$$P(Klasse = gut | y > 3) = \frac{P(y > 3 | Klasse = gut) \cdot P(Klasse = gut)}{P(y > 3)} = \frac{7/9 \cdot 9/13}{7/13} = 1$$

$$P(Klasse = gut | y \leq 3) = \frac{P(y \leq 3 | Klasse = gut) \cdot P(Klasse = gut)}{P(y \leq 3)} = \frac{2/9 \cdot 9/13}{6/13} = \frac{1}{3}$$

b) $P(y > 3 | Klasse = gut) = 7/9$

Aufgabe 7.3

a) 8 Ereignisse.

b) $P(Nied = trocken | Him = klar, Bar = steigt)$

$$= \frac{P(Nied = trocken, Him = klar, Bar = steigt)}{P(Him = klar, Bar = steigt)}$$

$$= \frac{0,4}{0,47} = 0,85$$

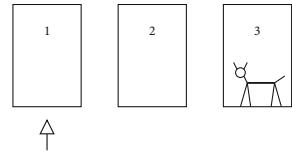
c) Fehlende Zeile in der Tabelle:

	bewölkt	fällt	regen	0,12
--	---------	-------	-------	------

$$P(Nied = regen | Him = bewölkt) = \frac{P(Nied = regen, Him = bewölkt)}{P(Him = bewölkt)} = \frac{0,23}{0,35} = 0,66$$

d) Die fehlende Wahrscheinlichkeitsmasse ist 0,15. Das Indifferenzprinzip (Definition 7.5) fordert nun, den beiden fehlenden Zeilen symmetrisch je den Wert 0,075 zuzuweisen.

Aufgabe 7.4 Der Kandidat hat sich zuerst für Tür 1 entschieden. Dann öffnet der Moderator Tür 3. Wir führen die Abkürzungen A_i für „Auto steht (vor Beginn des Experiments) hinter Tür i “ und M_i für „Moderator öffnet Tür i “ ein. Es gilt $P(A_1) = P(A_2) = P(A_3) = 1/3$ und wir berechnen



$$\begin{aligned} P(A_1 | M_3) &= \frac{P(M_3 | A_1)P(A_1)}{P(M_3)} \\ &= \frac{P(M_3 | A_1)P(A_1)}{P(M_3 | A_1)P(A_1) + P(M_3 | A_2)P(A_2) + P(M_3 | A_3)P(A_3)} \\ &= \frac{1/2 \cdot 1/3}{1/2 \cdot 1/3 + 1 \cdot 1/3 + 0 \cdot 1/3} = \frac{1/2 \cdot 1/3}{1/2} = 1/3 \\ P(A_2 | M_3) &= \frac{P(M_3 | A_2)P(A_2)}{P(M_3)} = \frac{1 \cdot 1/3}{1/2} = 2/3 \end{aligned}$$

Damit ist klar, dass das Wechseln der Tür besser ist.

Aufgabe 7.5 Die Lagrangefunktion lautet $L = -\sum_{i=1}^n p_i \ln p_i + \lambda(\sum_{i=1}^n p_i - 1)$. Nullsetzen der partiellen Ableitungen nach p_i und p_j ergibt

$$\frac{\partial L}{\partial p_i} = -\ln p_i - 1 + \lambda = 0 \quad \text{und} \quad \frac{\partial L}{\partial p_j} = -\ln p_j - 1 + \lambda = 0$$

Subtraktion dieser beiden Gleichungen führt auf $p_i = p_j$ für alle $i, j \in \{1, \dots, n\}$. Also gilt $p_1 = p_2 = \dots = p_n = 1/n$.

Aufgabe 7.6 PIT-Eingabe-Datei:

```
var A{t,f}, B{t,f};
P([A=t]) = 0{,}5;
P([B=t] | [A=t]) = 0{,}94;
QP([B=t]);
```

PIT-Ausgabe:

```
Reporting State of Queries
Nr  Truthvalue  Probability  Query
1   UNSPECIFIED  7{,}200e-01  QP([B=t]);
```

Da PIT numerisch arbeitet, kann es nicht symbolisch mit den Parametern α und β rechnen, sondern nur mit konkreten Zahlenwerten.

Aufgabe 7.7 Sei: $p_1 = P(A, B)$, $p_2 = P(A, \neg B)$, $p_3 = P(\neg A, B)$, $p_4 = P(\neg A, \neg B)$

Die Nebenbedingungen sind:

$$p_1 + p_2 = \alpha \quad (11.1)$$

$$p_4 = 1 - \beta \quad (11.2)$$

$$p_1 + p_2 + p_3 + p_4 = 1 \quad (11.3)$$

Daraus folgt: $p_3 = \beta - \alpha$. Aus (11.1) folgern wir aufgrund der Indifferenz $p_1 = p_2 = \alpha/2$. Also gilt $P(B) = p_1 + p_3 = \alpha/2 + \beta - \alpha = \beta - \alpha/2 = P(A \vee B) - 1/2P(A)$. Das PIT-Programm dazu lautet:

```
var A{t,f}, B{t,f};  
P([A=t]) = 0{}, 6;  
P([B=t] OR [A=t]) = 0{}, 94;  
QP([B=t]);
```

Aufgabe 7.8 Die Lagrangefunktion lautet

$$L = - \sum_{i=1}^4 p_i \ln p_i + \lambda_1(p_1 + p_2 - \alpha) + \lambda_2(p_1 + p_3 - \gamma) + \lambda_3(p_1 + p_2 + p_3 + p_4 - 1). \quad (11.4)$$

Partiell abgeleitet nach p_1, p_2, p_3, p_4 und λ erhalten wir

$$\frac{\partial L}{\partial p_1} = -\ln p_1 - 1 + \lambda_1 + \lambda_2 + \lambda_3 = 0 \quad (11.5)$$

$$\frac{\partial L}{\partial p_2} = -\ln p_2 - 1 + \lambda_1 + \lambda_3 = 0 \quad (11.6)$$

$$\frac{\partial L}{\partial p_3} = -\ln p_3 - 1 + \lambda_2 + \lambda_3 = 0 \quad (11.7)$$

$$\frac{\partial L}{\partial p_4} = -\ln p_4 - 1 + \lambda_3 = 0 \quad (11.8)$$

und berechnen

$$(11.5) - (11.6): \quad \ln p_2 - \ln p_1 + \lambda_2 = 0 \quad (11.9)$$

$$(11.7) - (11.8): \quad \ln p_4 - \ln p_3 + \lambda_2 = 0, \quad (11.10)$$

woraus $p_3/p_4 = p_1/p_2$ folgt, was wir sofort in (7.12) einsetzen:

$$\begin{aligned} & \frac{p_2 p_3}{p_4} + p_2 + p_3 + p_4 = 1 \\ \Leftrightarrow & p_2 \left(1 + \frac{p_3}{p_4}\right) + p_3 + p_4 = 1 \\ (7.10) - (7.11): & \quad p_2 = p_3 + \alpha - \gamma, \end{aligned} \quad (11.11)$$

was für p_2 eingesetzt

$$(p_3 + \alpha - \gamma) \left(1 + \frac{p_3}{p_4} \right) + p_3 + p_4 = 1 \quad (11.12)$$

ergibt.

$$(7.10) \text{ in } (7.12): \quad \alpha + p_3 + p_4 = 1 \quad (11.13)$$

Damit eliminieren wir p_4 in (11.12), was zu

$$(p_3 + \alpha - \gamma) \left(1 + \frac{p_3}{1 - \alpha - p_3} \right) + 1 - \alpha = 1 \quad (11.14)$$

$$\Leftrightarrow (p_3 + \alpha - \gamma)(1 - \alpha - p_3 + p_3) = \alpha(1 - \alpha - p_3) \quad (11.15)$$

$$\Leftrightarrow (p_3 + \alpha - \gamma)(1 - \alpha) = \alpha(1 - \alpha - p_3) \quad (11.16)$$

$$\Leftrightarrow p_3 + \alpha - \gamma - \alpha p_3 - \alpha^2 + \alpha \gamma = \alpha - \alpha^2 - \alpha p_3 \quad (11.17)$$

$$\Leftrightarrow p_3 = \gamma(1 - \alpha) \quad (11.18)$$

führt. Mit (11.13) ergibt sich daraus $p_4 = (1 - \alpha)(1 - \gamma)$ und mit (11.11) erhält man $p_2 = \alpha(1 - \gamma)$ und $p_1 = \alpha\gamma$.

Aufgabe 7.9

a)

$$M = \begin{pmatrix} 0 & 1000 \\ 10 & 0 \end{pmatrix}$$

b)

$$M \cdot \begin{pmatrix} p \\ 1-p \end{pmatrix} = \begin{pmatrix} 1000 \cdot (1-p) \\ 10 \cdot p \end{pmatrix}$$

Da die Entscheidung für *Löschen* oder *Lesen* durch Bildung des Minimums der beiden Werte getroffen wird, genügt der Vergleich der beiden Werte: $1000(1 - p) < 10p$, was zu $p > 0,99$ führt. Allgemein berechnet man für eine Matrix $M = \begin{pmatrix} 0 & k \\ 1 & 0 \end{pmatrix}$ die Schwelle $k/(k+1)$.

Aufgabe 7.10

- a) Da A und B unabhängig sind, gilt $P(A | B) = P(A) = 0,2$
- b) Durch Anwendung der Konditionierungsregel (Abschn. 7.4.7) erhält man

$$P(C | A) = P(C | A, B)P(B) + P(C | A, \neg B)P(\neg B) = 0,2 \cdot 0,9 + 0,1 \cdot 0,1 = 0,19$$

Aufgabe 7.11

a) $P(Al) = P(Al, Ein, Erd) + P(Al, \neg Ein, Erd) + P(Al, Ein, \neg Erd) + P(Al, \neg Ein, \neg Erd)$

$$= P(Al|Ein, Erd)P(Ein, Erd) + P(Al|\neg Ein, Erd)P(\neg Ein, Erd)$$

$$+ P(Al|Ein, \neg Erd)P(Ein, \neg Erd) + P(Al|\neg Ein, \neg Erd)P(\neg Ein, \neg Erd)$$

$$= 0,95 \cdot 0,001 \cdot 0,002 + 0,29 \cdot 0,999 \cdot 0,002 + 0,94 \cdot 0,001 \cdot 0,998$$

$$+ 0,001 \cdot 0,999 \cdot 0,998$$

$$= 0,00252$$

$$P(J) = P(J, Al) + P(J, \neg Al) = P(J|Al)P(Al) + P(J|\neg Al)P(\neg Al)$$

$$= 0,9 \cdot 0,0025 + 0,05 \cdot (1 - 0,0025) = 0,052$$

$$P(M) = P(M, Al) + P(M, \neg Al) = P(M|Al)P(Al) + P(M|\neg Al)P(\neg Al)$$

$$= 0,7 \cdot 0,0025 + 0,01 \cdot (1 - 0,0025) = 0,0117$$

b) $P(Al|Ein) = P(Al|Ein, Erd)P(Erd) + P(Al|Ein, \neg Erd)P(\neg Erd)$

$$= 0,95 \cdot 0,002 + 0,94 \cdot 0,998 = 0,94002$$

$$P(M|Ein) = P(M, Ein)/P(Ein) = [P(M, Al, Ein) + P(M, \neg Al, Ein)]/P(Ein)$$

$$= [P(M|Al)P(Al|Ein)P(Ein) + P(M|\neg Al)P(\neg Al|Ein)P(Ein)]/P(Ein)$$

$$= P(M|Al)P(Al|Ein) + P(M|\neg Al)P(\neg Al|Ein)$$

$$= 0,7 \cdot 0,94002 + 0,01 \cdot 0,05998 = 0,659$$

c) $P(Ein|M) = \frac{P(M|Ein)P(Ein)}{P(M)} = \frac{0,659 \cdot 0,001}{0,0117} = 0,056$

d) $P(Al|J, M) = \frac{P(Al, J, M)}{P(J, M)} = \frac{P(Al, J, M)}{P(Al, J, M) + P(\neg Al, J, M)} = \frac{1}{1 + \frac{P(\neg Al, J, M)}{P(Al, J, M)}}$

$$= \frac{1}{1 + \frac{P(J|\neg Al)P(M|\neg Al)P(\neg Al)}{P(J|Al)P(M|Al)P(Al)}} = \frac{1}{1 + \frac{0,05 \cdot 0,01 \cdot 0,9975}{0,9 \cdot 0,7 \cdot 0,0025}} = 0,761$$

$$P(J, M|Ein) = P(J, M|Al)P(Al|Ein) + P(J, M|\neg Al)P(\neg Al|Ein)$$

$$= P(J|Al)P(M|Al)P(Al|Ein) + P(J|\neg Al)P(M|\neg Al)P(\neg Al|Ein)$$

$$= 0,9 \cdot 0,7 \cdot 0,94 + 0,05 \cdot 0,01 \cdot 0,06 = 0,5922$$

$$P(Al|\neg Ein) = P(Al|\neg Ein, Erd)P(Erd) + P(Al|\neg Ein, \neg Erd)P(\neg Erd)$$

$$= 0,29 \cdot 0,002 + 0,001 \cdot 0,998 = 0,00158$$

$$P(J, M|\neg Ein) = P(J, M|Al)P(Al|\neg Ein) + P(J, M|\neg Al)P(\neg Al|\neg Ein)$$

$$= P(J|Al)P(M|Al)P(Al|\neg Ein) + P(J|\neg Al)P(M|\neg Al)P(\neg Al|\neg Ein)$$

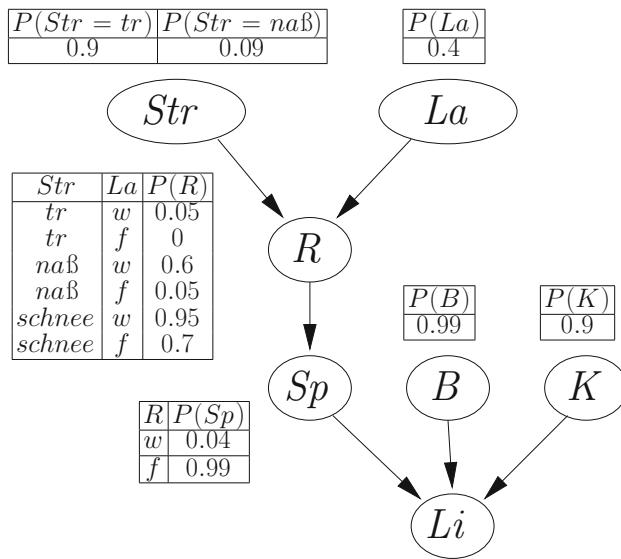
$$= 0,9 \cdot 0,7 \cdot 0,00158 + 0,05 \cdot 0,01 \cdot 0,998 = 0,00149$$

$$P(J, M) = P(J, M|Ein)P(Ein) + P(J, M|\neg Ein)P(\neg Ein)$$

$$= 0,5922 \cdot 0,001 + 0,00149 \cdot 0,999 = 0,00208$$

$$P(Ein|J, M) = \frac{P(J, M|Ein)P(Ein)}{P(J, M)} = 0,5922 \cdot 0,001 / 0,00208 = 0,284$$

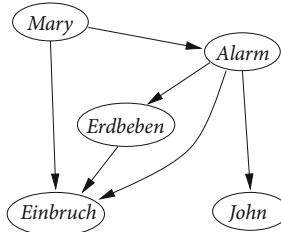
Abb. 11.1 Bayes-Netz für die Fahrradlichtanlage. Die CPT für Li ist in der Aufgabenstellung angegeben



e) $P(J)P(M) = 0,052 \cdot 0,0117 = 0,00061$, aber $P(J, M) = 0,00208$ (siehe oben). Also $P(J)P(M) \neq P(J, M)$.

f) Siehe Abschn. 7.4.5.

g)



Es gibt auch noch andere Lösungen.

h) Es ändert sich nur die CPT des Alarm-Knotens. Alle anderen Knoten sind unabhängig von Erdbeben oder unabhängig von Erdbeben gegeben Alarm.

Ein	P(Al)
w	0,94
f	0,0016

Aufgabe 7.12

a), b) Siehe Abb. 11.1

c) Die Kante (Str, Li) fehlt, denn Li und Str sind bedingt unabhängig, gegeben Sp , d. h. es gilt $P(Li | Sp, Str) = P(Li | Sp)$, denn der Straßenzustand hat keinen direkten Einfluss auf das Licht, sondern nur über den Dynamo und die von ihm erzeugte Spannung. (Die bed. Unabh. von Li und Str gegeben Sp kann hier wegen nicht vorhandener Daten für

$P(Li | Sp, Str)$ sowie $P(Li | Sp)$ nicht durch Berechnung gezeigt werden, sondern nur durch Argumentation.)

- d) Bei den angegebenen CPTs ergibt sich

$$\begin{aligned} P(R | Str) &= P(R | Str, La)P(La) + P(R | Str, \neg La)P(\neg La) \\ &= 0,95 \cdot 0,4 + 0,7 \cdot 0,6 = 0,8 \\ P(Sp | Str) &= P(Sp | R)P(R | Str) + P(Sp | \neg R)P(\neg R | Str) \\ &= 0,04 \cdot 0,8 + 0,99 \cdot 0,2 = 0,23. \end{aligned}$$

11.8 Maschinelles Lernen und Data Mining

Aufgabe 8.1

- a) Der Agent ist eine Funktion A , die einen Vektor $(t_1, t_2, t_3, d_1, d_2, d_3, f_1, f_2, f_3)$ bestehend aus jeweils drei Temperatur-, Druck- und Feuchtigkeitswerten abbildet auf einen Klassenwert $k \in \{\text{sonnig, bewölkt, regnerisch}\}$.
 b) Die Trainingsdatadatei besteht aus je einer Zeile der Form

$$t_{i1}, t_{i2}, t_{i3}, d_{i1}, d_{i2}, d_{i3}, f_{i1}, f_{i2}, f_{i3}, k_i$$

für jedes einzelne Trainingsdatum. Der Index i läuft über alle Trainingsdaten. Eine konkrete Datei könnte zum Beispiel so beginnen:

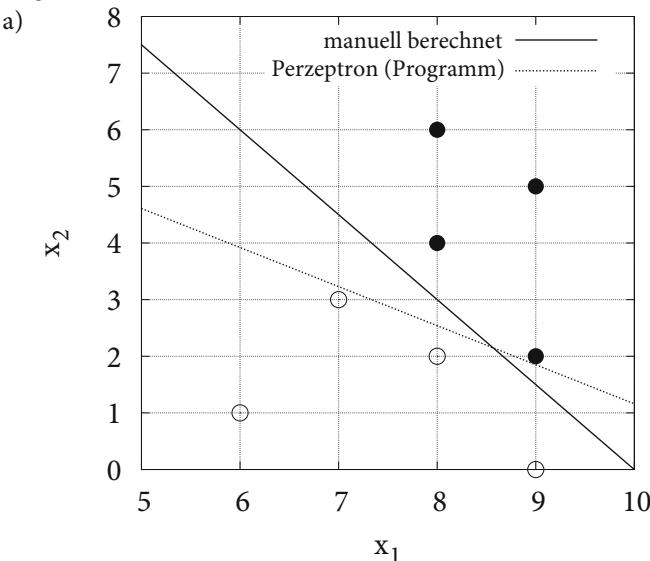
17, 17, 17, 980, 983, 986, 63, 61, 50, sonnig
 20, 22, 25, 990, 1014, 1053, 65, 66, 69, sonnig
 20, 22, 18, 990, 979, 929, 52, 60, 61, regnerisch

Aufgabe 8.2 Die Symmetrie erkennt man direkt anhand der Definition Korrelationskoeffizienten, bzw. der Kovarianz. Vertauschen von i und j ändert den Wert nicht. Für die Diagonalelemente berechnet man

$$K_{ii} = \frac{\sigma_{ii}}{s_i \cdot s_i} = \frac{\sum_{p=1}^N (x_i^p - \bar{x}_i)(x_i^p - \bar{x}_i)}{\sum_{p=1}^N (x_i^p - \bar{x}_i)^2} = 1.$$

Aufgabe 8.3 Die Folge der Werte für w ist:

$$(1,1), (2,2, -0,4), (1,8, 0,6), (1,4, 1,6), (2,6, 0,2), (2,2, 1,2)$$

Aufgabe 8.4

b) Gerade in Zeichnung einzeichnen ergibt:

$$1,5x_1 + x_2 - 15 > 0.$$

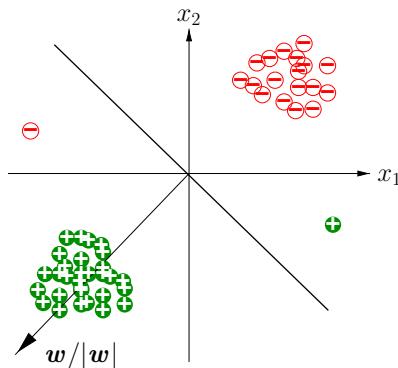
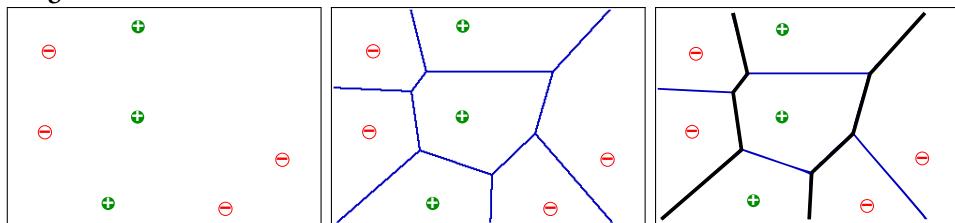
Perceptron-Lernalgorithmus mit Startvektor $w = (1, 1, 1)$ liefert nach 442 Iterationen:

$$w = (11, 16, -129).$$

Das entspricht: $0,69x_1 + x_2 - 8,06 > 0$

Aufgabe 8.5

- a) Der Vektor $\sum_{x \in M_+} x$ zeigt in eine „mittlere“ Richtung aller positiven Punkte und $\sum_{x \in M_-} x$ zeigt in eine „mittlere“ Richtung aller negativen Punkte. Die Differenz dieser Vektoren zeigt von den negativen in Richtung der positiven Punkte. Die Trenngerade steht dann senkrecht auf dieser Differenz.
- b) Die Punktewolken der positiven und negativen Punkte dominieren bei der Berechnung von w . Die beiden Ausreißer spielen hierbei fast keine Rolle. Für die Bestimmung der Trenngeraden sind sie aber wichtig.

**Aufgabe 8.6****Aufgabe 8.7**

- a) Nächster Nachbar ist $(8, 4)$, also Klasse 1.
- b) $k = 2$: Klasse undefiniert, da 1 mal Klasse 1 und 1 mal 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 machen zu können, muss man annehmen, dass die Punkte gleichmäßig im Merkmalsraum verteilt sind, d. h. die Zahl der Punkte pro Fläche ist überall etwa gleich. Nun berechnen wir die Fläche A_d eines schmalen Rings der Breite Δd im Abstand d um den Punkt x :

$$A_d = \pi(d + \Delta d)^2 - \pi d^2 = \pi(d^2 + 2d\Delta d + \Delta d^2) - \pi d^2 \approx 2\pi d \Delta d$$

Das gesamte Gewicht aller Punkte im Ring der Breite Δd im Abstand d ist damit proportional zu $dw_i = d/(1 + \alpha d^2) \approx 1/(\alpha d)$ für $d \rightarrow \infty$.

- b) Bei dieser Gewichtung wäre das gesamte Gewicht aller Punkte im Ring der Breite Δd im Abstand d proportional zu $dw'_i = d/d = 1$. Damit hätte also jeder Ring der Breite Δd , unabhängig von seinem Abstand zum Punkt x gleiches Gewicht. Das ist sicher nicht sinnvoll, denn die nähere Umgebung ist für die Approximation am wichtigsten.

Aufgabe 8.10 $\lim_{x \rightarrow 0} x \log_2 x = \lim_{x \rightarrow 0} \frac{\log_2 x}{1/x} = \lim_{x \rightarrow 0} \frac{1/(x \ln 2)}{-1/x^2} = \lim_{x \rightarrow 0} \frac{-x}{\ln 2} = 0$, wobei für die zweite Gleichung die Regel von de l'Hospital verwendet wurde.

Aufgabe 8.11 a) 0 b) 1 c) 1,5 d) 1,875 e) 2,32 f) 2

Aufgabe 8.12

- a) Aus $\log_2 x = \ln x / \ln 2$ folgt $c = 1 / \ln 2 \approx 1,44$.
- b) Da sich die beiden Entropiefunktionen nur um einen konstanten Faktor unterscheiden, ändert sich die Lage des Entropiemaximums nicht. Auch Extrema unter Nebenbedingungen ändern ihre Lage nicht. Also stört der Faktor c bei der MaxEnt-Methode nicht. Beim Lernen von Entscheidungsbäumen werden die Informationsgewinne verschiedener Attribute verglichen. Da hier nur die Ordnung eine Rolle spielt, nicht aber der absolute Wert, stört auch hier der Faktor c nicht.

Aufgabe 8.13

- a) Für das erste Attribut berechnet man:

$$\begin{aligned} G(D, x_1) &= H(D) - \sum_{i=6}^9 \frac{|D_{x_1=i}|}{|D|} H(D_{x_1=i}) \\ &= 1 - \left(\frac{1}{8} H(D_{x_1=6}) + \frac{1}{8} H(D_{x_1=7}) + \frac{3}{8} H(D_{x_1=8}) + \frac{3}{8} H(D_{x_1=9}) \right) \\ &= 1 - \left(0 + 0 + \frac{3}{8} \cdot 0,918 + \frac{3}{8} \cdot 0,918 \right) = 0,311 \end{aligned}$$

$G(D, x_2) = 0,75$, also wird x_2 gewählt. Für $x_2 = 0, 1, 3, 4, 5, 6$ ist die Entscheidung klar. Für $x_2 = 2$ wird x_1 gewählt. Der Baum hat dann die Gestalt

```

x2 = 0: 0 (1/0)
x2 = 1: 0 (1/0)
x2 = 2:
| x1 = 6: 0 (0/0)
| x1 = 7: 0 (0/0)
| x1 = 8: 0 (1/0)
| x1 = 9: 1 (1/0)
x2 = 3: 0 (1/0)
x2 = 4: 1 (1/0)
x2 = 5: 1 (1/0)
x2 = 6: 1 (1/0)

```

- b) Informationsgewinn für das stetige Attribut x_2 als Wurzelknoten:

Schwelle Θ	0	1	2	3	4	5
$G(D, x_2 \leq \Theta)$	0,138	0,311	0,189	0,549	0,311	0,138

Da $G(D, x_2 \leq 3) = 0,549 > 0,311 = G(D, x_1)$, wird $x_2 \leq 3$ gewählt. Für $x_2 \leq 3$ ist die Klassifikation nicht eindeutig. Wir rechnen $G(D_{x_2 \leq 3}, x_1) = 0,322$, $G(D_{x_2 \leq 3}, x_2 \leq 0) = 0,073$, $G(D_{x_2 \leq 3}, x_2 \leq 1) = 0,17$, $G(D_{x_2 \leq 3}, x_2 \leq 2) = 0,073$. Also wird x_1 gewählt. Für $x_1 = 9$ ist die Klassifikation nicht eindeutig. Hier ist die Entscheidung klar $G(D_{(x_2 \leq 3, x_1=9)}, x_1) = 0$, $G(D_{(x_2 \leq 3, x_1=9)}, x_2 \leq 1) = 1$, es wird $x_2 \leq 1$ gewählt und der Baum hat wieder 100 % Korrektheit.

Baum:

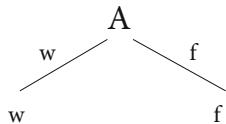
```

x2 <= 3 :
| x1 = 6: 0 (1/0)
| x1 = 7: 0 (0/0)
| x1 = 8: 0 (1/0)
| x1 = 9:
|   | x2 <= 1: 0 (1/0)
|   | x2 > 1: 1 (1/0)
x2 > 3: 1 (3/0)

```

Aufgabe 8.14

- a) 100 % Korrektheit auf den Trainingsdaten, 75 % Korrektheit auf den Testdaten, 80 % Korrektheit gesamt
- b) $(A \wedge \neg C) \vee (\neg A \wedge B)$
- c)



66,6 % Korrektheit auf den Trainingsdaten

100 % Korrektheit auf den Testdaten

80 % Korrektheit gesamt

Aufgabe 8.15

- a) Gleichung (8.7) zur Berechnung des Informationsgewinns eines Attributs A lautet

$$\text{InfGewinn}(D, A) = H(D) - \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i),$$

wobei n die Zahl der Werte des aktuell betrachteten Attributs ist. Wird nun das Attribut A irgendwo im Unterbaum als Nachfolger des Wertes a_j wieder getestet, so kommt in der Datenmenge D_j und jeder Teilmenge D' davon nur noch der Wert $A = a_j$ vor. Also ist $D' = D'_j$ und für alle $k \neq j$ gilt $|D'_k| = 0$ und wir erhalten

$$\text{InfGewinn}(D', A) = H(D') - \sum_{i=1}^n \frac{|D'_i|}{|D'|} H(D'_i) = H(D'_j) - \frac{|D'_j|}{|D'_j|} H(D'_j) = 0.$$

Der Informationsgewinn des wiederholten Attributs ist also null, weshalb es nicht mehr verwendet wird.

- b) Aus jedem stetigen Attribut A wird ein binäres Attribut $A > \Theta_{D,A}$ erzeugt. Wird nun weiter unten im Baum das Attribut A nochmal diskretisiert mit einer anderen Schwellen $\Theta_{D',A}$, so unterscheidet sich das Attribut $A > \Theta_{D',A}$ von $A > \Theta_{D,A}$. Wenn es dann höheren Informationsgewinn als alle anderen Attribute hat, wird es im Baum verwendet. Das heißt aber auch, dass bei mehrmaligem Vorkommen eines stetigen Attributs die Schwellen verschieden sein müssen.

Aufgabe 8.16

- a) $P(Him = klar) = 0,65$
 $P(Bar = steigt) = 0,67$

Him	Bar	$P(Nied = trocken Him, Bar)$
klar	steigt	0,85
klar	fällt	0,44
bewölkt	steigt	0,45
bewölkt	fällt	0,2

- b) $P(Him = klar) = 0,65$

Bar	$P(Nied = trocken Bar)$	Him	$P(Bar = steigt Him)$
steigt	0,73	klar	0,72
fällt	0,33	bewölkt	0,57

- c) Die benötigten CPTs für $P(Nied|Him, Bar)$ und $P(Bar|Him)$, sowie $P(Him)$ sind schon bekannt.

d) $\mathbf{P}_a = (0,37, 0,065, 0,095, 0,12, 0,11, 0,13, 0,023, 0,092)$
 $\mathbf{P}_b = (0,34, 0,13, 0,06, 0,12, 0,15, 0,054, 0,05, 0,1)$
 $\mathbf{P} = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$ (Original-Verteilung)

quadratische Abstände: $d_q(\mathbf{P}_a, \mathbf{P}) = 0,0029$, $d_q(\mathbf{P}_b, \mathbf{P}) = 0,014$

Kullback-Leibler-Abst.: $d_k(\mathbf{P}_a, \mathbf{P}) = 0,017$, $d_k(\mathbf{P}_b, \mathbf{P}) = 0,09$

Beide Abstandsmaße zeigen, dass das Netz a die Verteilung besser approximiert als Netz b. Das heißt unter anderem, dass die Annahme, *Nied* und *Him* sind bedingt unabhängig gegeben *Bar* weniger gut zutrifft als die Annahme, *Him* und *Bar* sind unabhängig.

- e) $\mathbf{P}_c = (0,4, 0,07, 0,08, 0,1, 0,09, 0,11, 0,03, 0,12)$. Diese Verteilung ist exakt gleich der Originalverteilung \mathbf{P} . Dies ist nicht überraschend, denn in dem Netz fehlt keine Kante. Das heißt, es wurde keine Unabhängigkeit angenommen.

Aufgabe 8.17 Dass Score und Perzeptron äquivalent sind, erkennt man sofort durch Vergleich der Definitionen. Nun zur Äquivalenz zu Naive-Bayes: Zuerst stellen wir fest, dass $P(K|S_1, \dots, S_n) > 1/2$ äquivalent ist zu $P(K|S_1, \dots, S_n) > P(\neg K|S_1, \dots, S_n)$, denn $P(\neg K|S_1, \dots, S_n) > 1 - P(K|S_1, \dots, S_n)$. Es handelt sich hier also tatsächlich um einen binären Naive-Bayes-Klassifizierer.

Wir logarithmieren die Naive-Bayes-Formel

$$P(K|S_1, \dots, S_n) = \frac{P(S_1|K) \cdot \dots \cdot P(S_n|K) \cdot P(K)}{P(S_1, \dots, S_n)},$$

und erhalten

$$\log P(K|S_1, \dots, S_n) = \log P(S_1|K) + \dots + \log P(S_n|K) + \log P(K) - \log P(S_1, \dots, S_n). \quad (11.19)$$

Um einen Score zu erhalten, müssen wir die Variablen S_1, \dots, S_n als numerische Variablen mit den Werten 1 und 0 interpretieren. Man erkennt leicht, dass

$$\log P(S_i|K) = (\log P(S_i=1|K) - \log P(S_i=0|K))S_i + \log P(S_i=0|K).$$

Daraus folgt dann

$$\sum_{i=1}^n \log P(S_i|K) = \sum_{i=1}^n (\log P(S_i=1|K) - \log P(S_i=0|K))S_i + \sum_{i=1}^n \log P(S_i=0|K).$$

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

$$\sum_{i=1}^n \log P(S_i|K) = \sum_{i=1}^n w_i S_i + c.$$

Eingesetzt in (11.19) 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-Klassifizierers

$$\log P(K|S_1, \dots, S_n) > \log(1/2)$$

sein. Also muss

$$\sum_{i=1}^n w_i S_i + c + \log P(K) - \log P(S_1, \dots, S_n) > \log(1/2)$$

sein, bzw.

$$\sum_{i=1}^n w_i S_i > \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n),$$

womit wir einen Score mit der Schwelle $\Theta = \log 1/2 - c - \log P(K) + \log P(S_1, \dots, S_n)$ definiert haben. Da sich alle Umformungen rückgängig machen lassen, kann man umgekehrt auch jeden Score in einen Bayes-Klassifizierer transformieren, womit die Äquivalenz gezeigt wäre.

Aufgabe 8.18 Logarithmieren von (8.10) führt zu

$$\log P(I | s_1, \dots, s_n) = \log c + \log P(I) + \sum_{i=1}^l n_i \log P(w_i | I).$$

Dadurch werden aus sehr kleinen positiven Werten moderate negative Zahlen. Da die Logarithmusfunktion monoton wächst, wird zur Ermittlung der Klasse maximiert nach der Vorschrift

$$I_{\text{Naive-Bayes}} = \underset{I \in \{w, f\}}{\operatorname{argmax}} \log P(I | s_1, \dots, s_n).$$

Nachteil dieser Methode ist die bei großen Texten etwas längere Rechenzeit in der Lernphase. Bei der Klassifikation steigt die Zeit nicht, denn die Werte $\log P(I | s_1, \dots, s_n)$ können beim Lernen gespeichert werden.

Aufgabe 8.20 Sei also f streng monoton wachsend, das heißt $\forall x, y \ x < y \Rightarrow f(x) < f(y)$. Wenn nun $d_1(s, t) < d_1(u, v)$ ist, dann ist offenbar

$$d_2(s, t) = f(d_1(s, t)) < f(d_1(u, v)) = d_2(u, v).$$

Da die Umkehrfunktion von f auch streng monoton ist, gilt die Umkehrung, das heißt $d_2(s, t) < d_2(u, v) \Rightarrow d_1(s, t) < d_1(u, v)$. Damit ist $d_2(s, t) < d_2(u, v) \Leftrightarrow d_1(s, t) < d_1(u, v)$ gezeigt.

Aufgabe 8.21 $x_1 x_2 = 2$, und damit $d_s(x_1, x_2) = \frac{\sqrt{20 \cdot 24}}{2} = 10,95$

$x_2 x_3 = 4$, und damit $d_s(x_2, x_3) = \frac{\sqrt{24 \cdot 18}}{4} = 5,20$

$x_1 x_3 = 2$, und damit $d_s(x_1, x_3) = \frac{\sqrt{20 \cdot 18}}{2} = 9,49$

Es scheint so, dass sich trotz Übersetzung ins Deutsche die beiden Sätze von Turing ähnlicher sind als der Satz des Autors und Turings Sätze.

Aufgabe 8.22 Hilfe bei Problemen mit KNIME: <http://www.knime.org/forum>

11.9 Neuronale Netze

Aufgabe 9.1 Zu zeigen ist, dass $f(\Theta + x) + f(\Theta - x) = 1$.

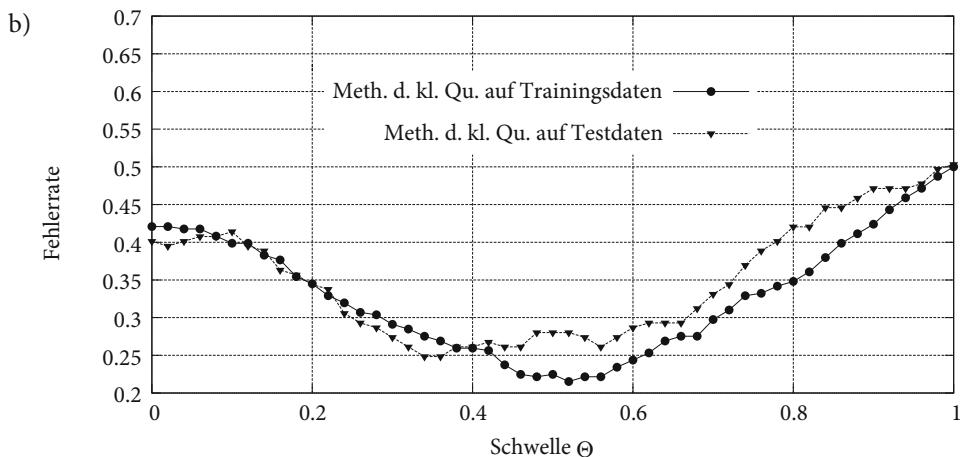
$$f(\Theta + x) = \frac{1}{1 + e^{-\frac{x}{T}}} = \frac{e^{\frac{x}{T}}}{1 + e^{\frac{x}{T}}}, \quad f(\Theta - x) = \frac{1}{1 + e^{\frac{x}{T}}}, \quad f(\Theta + x) + f(\Theta - x) = 1$$

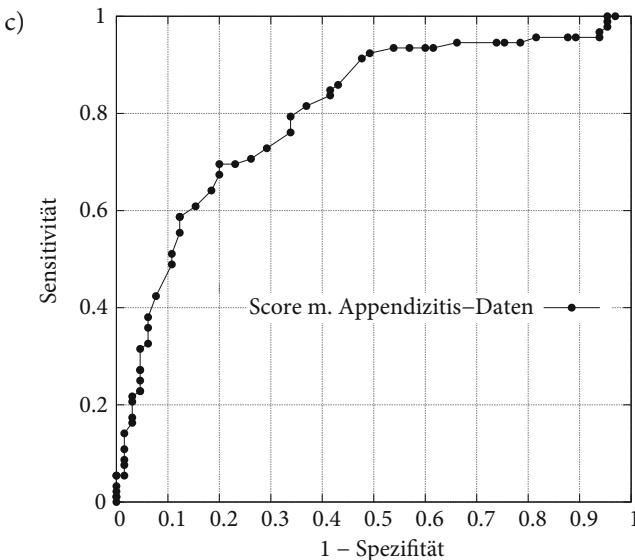
Aufgabe 9.3 Jedes der im Netz gespeicherten Muster hat eine Größe von n Bit. Das Netz besitzt insgesamt $n(n-1)/2$ Gewichte. Wenn wir pro Gewicht 16 Bit reservieren und als gleich großen binären Speicher einen der Größe $16n(n-1)/2$ definieren, so kann dieser offenbar $N = 8n(n-1)/n = 4(n-1)$ Muster der Größe n Bit speichern. Für große n erhalten wir als Grenzwert $N = 4n$. Wenn wir, wie in (9.11), den Quotienten α aus der Zahl der speicherbaren Bits und der Zahl verfügbarer Speicherzellen bilden, so erhalten wir für den Listenspeicher den Wert 1 und für das Hopfield-Netz den Wert $\alpha = 0,146n^2/(16n(n-1)/2) \approx 0,018$. Der klassische Speicher hat also (bei 16 Bit pro Gewicht) eine etwa 55 mal höhere Kapazität.

Aufgabe 9.4

- a) Mathematica-Programm für die Methode der kleinsten Quadrate:

```
LeastSq[q_, a_] := Module[{Nq, Na, m, A, b, w},
  Nq = Length[q]; m = Length[q[[1]]]; Na = Length[a];
  If[Nq != Na, Print["Length[q] != Length[a]"]; Exit, 0];
  A = Table[N[Sum[q[[p, i]] q[[p, j]], {p, 1, Nq}]], {i, 1, m}, {j, 1, m}];
  b = Table[N[Sum[a[[p]] q[[p, j]], {p, 1, Nq}]], {j, 1, m}];
  w = LinearSolve[A, b];
]
LeastSq::usage = "LeastSq[x,y,f] berechnet aus den Anfragevektoren q[[1]], ..., q[[m]] eine Tabelle aller Koeffizienten w[[i]] fuer eine lineare Abbildung f[x] = Sum[w[[i]] x[[i]], {i,1,m}] mit f[q[[p]]] = a[[p]]."
```





Aufgabe 9.6 a) Lernen klappt ohne Fehler. b) Lernen geht 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 gilt $f(g(x + y)) = f(g(x) + g(y)) = f(g(x)) + f(g(y))$ und $f(g(kx)) = f(kg(x)) = kf(g(x))$. Also ist hintereinander Ausführen von linearen Abbildungen eine lineare Abbildung.
- b) Wir betrachten zwei beliebige Ausgabeneuronen j und k . Jedes der beiden stehe für eine Klasse. Die Klassifikation erfolgt durch Maximumbildung der beiden Aktivierungen. Seien $net_j = \sum_i w_{ji}x_i$ und $net_k = \sum_i w_{ki}x_i$ die gewichteten Summen von Werten, die bei den Neuronen j und k ankommen. Sei außerdem $net_j > net_k$. Ohne Aktivierungsfunktion wird hier Klasse j ausgegeben. Wird nun auf die Ergebnisse eine streng monotone Aktivierungsfunktion f angewendet, so ändert sich nichts am Ergebnis, denn auf Grund der strengen Monotonie gilt $f(net_j) > f(net_k)$.

Aufgabe 9.8 $f_1(x_1, x_2) = x_1^2$, $f_2(x_1, x_2) = x_2^2$. Die Trenngerade im transformierten Raum hat dann die Gleichung $y_1 + y_2 = 1$.

11.10 Lernen durch Verstärkung

Aufgabe 10.1 a) n^n b) $(n-1)^n$ c) $\curvearrowleft \bullet$ $\bullet \curvearrowright$ $\curvearrowleft \bullet \leftarrow \bullet$ $\bullet \rightarrow \bullet \curvearrowright$ $\bullet \leftrightarrow \bullet$



Aufgabe 10.2 Wert-Iteration mit zeilenweisem Aktualisieren der \hat{V} -Werte von links oben nach rechts unten ergibt

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

Aufgabe 10.3

c)

1	2	3	4	5	6
0	0	0	0	0	0
1	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
2	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
3	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
4	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0
5	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0
6	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0

6×6 Feedback bei ganz glattem Untergrund

1	2	3	4	5	6
0	0	0	0	0	0
1	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
2	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
3	0 0	0 0	0 0	0 0	0 0
0	0	0	0	0	0
4	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0
5	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0
6	1 -1	1 -1	1 -1	1 -1	1 -1
0	0	0	0	0	0

6×6 Feedback bei ganz glattem Untergrund

- d) Man sieht, dass je länger eine Strategie wird (d. h. je mehr Schritte z. B. ein Zyklus einer zyklischen Strategie hat), desto näher muss der Wert für γ bei 1 liegen, denn ein hoher Wert für γ ermöglicht ein längeres Gedächtnis. Umso langsamer konvergiert dann aber die Wert-Iteration.

Aufgabe 10.4 Der Wert $V^*(3, 3)$ rechts unten in der Matrix der Zustände wird folgendermaßen verändert:

$$V^*(3, 1) = 0,9 V^*(2, 1) = 0,9^2 V^*(2, 2) = 0,9^3 V^*(2, 3) = 0,9^4 V^*(3, 3). \quad (11.20)$$

Diese Gleichungskette folgt aus der Gleichung (10.7), denn für alle angegebenen Zustandsübergänge ist die maximale direkte Belohnung $r(s, a) = 0$ und es gilt

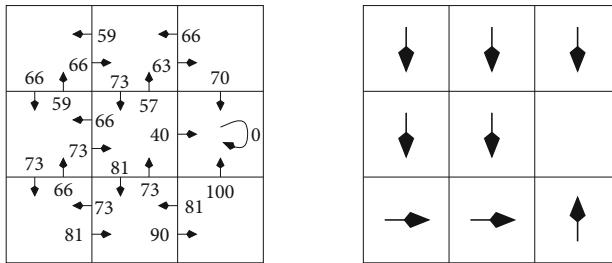
$$V^*(s) = \max_a [r(s, a) + \gamma V^*(\delta(s, a))] = \gamma V^*(\delta(s, a)) = 0,9 V^*(\delta(s, a)).$$

Aus (10.7) folgt auch $V^*(3, 2) = 1 + 0,9 V^*(3, 1)$, weil hier $r(s, a) = 1$ maximal ist. Analog gilt $V^*(3, 3) = 1 + 0,9 V^*(3, 2)$ und der Kreis schließt sich. Die beiden letzten Gleichungen zusammen ergeben $V^*(3, 3) = 1 + 0,9(1 + 0,9 V^*(3, 1))$. Aus (11.20) folgt $V^*(3, 1) = 0,9^4 V^*(3, 3)$. Dies in $V^*(3, 3)$ eingesetzt ergibt

$$V^*(3, 3) = 1 + 0,9(1 + 0,9^5 V^*(3, 3)),$$

woraus die Behauptung folgt.

Aufgabe 10.5 Stabile Q-Werte und eine optimale Strategie:



Aufgabe 10.6

a) Anzahl Zustände = ℓ^n , Anz. Aktionen pro Zustand = $\ell^n x - 1$. Anzahl Strategien = $(\ell^n)^{\ell^n} = \ell^{n\ell^n}$.

b)	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 10$
$n = 1$	1	4	27	256	10^{10}
$n = 2$	1	256	$3,9 \cdot 10^8$	$1,8 \cdot 10^{19}$	10^{200}
$n = 3$	1	$1,7 \cdot 10^7$	$4,4 \cdot 10^{38}$	$3,9 \cdot 10^{115}$	10^{3000}
$n = 4$	1	$1,8 \cdot 10^{19}$	$3,9 \cdot 10^{154}$	$3,2 \cdot 10^{616}$	$10^{40.000}$
$n = 8$	1	$3,2 \cdot 10^{616}$	$1,4 \cdot 10^{25.043}$	$6,7 \cdot 10^{315.652}$	$10^{800.000.000}$

c) Pro Zustand sind nun also $2n$ Aktionen möglich. Also gibt es $(2n)^{\ell^n}$ Strategien.

	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 10$
$n = 1$	2	4	8	16	1024
$n = 2$	4	256	$2,6 \cdot 10^9$	$4,3 \cdot 10^9$	$1,6 \cdot 10^{60}$
$n = 3$	6	$1,7 \cdot 10^6$	$1,0 \cdot 10^{21}$	$6,3 \cdot 10^{49}$	$1,4 \cdot 10^{778}$
$n = 4$	8	$2,8 \cdot 10^{14}$	$1,4 \cdot 10^{73}$	$1,6 \cdot 10^{231}$	$7,9 \cdot 10^{9030}$
$n = 8$	16	$1,8 \cdot 10^{308}$	$1,7 \cdot 10^{7900}$	$1,6 \cdot 10^{78.913}$	$1,8 \cdot 10^{120.411.998}$

- d) $10^{120.411.998}$ verschiedene Strategien können niemals kombinatorisch exploriert werden, auch nicht, wenn alle auf dieser Welt verfügbaren Rechner parallel daran arbeiten würden. Es werden also „intelligente“ Verfahren benötigt, um eine optimale oder auch nur eine fast optimale Strategie zu finden.

Literatur

- [Ada75] E.W. Adams. *The Logic of Conditionals*, Synthese Library Bd. 86. D. Reidel Publishing Company (1975).
- [Alp04] E. Alpaydin. *Introduction to Machine Learning*. MIT Press (2004).
- [APR90] J. Anderson, A. Pellionisz und E. Rosenfeld. *Neurocomputing (vol. 2): directions for research*. MIT Press, Cambridge, MA, USA (1990).
- [AR88] J. Anderson und E. Rosenfeld. *Neurocomputing: Foundations of Research*. MIT Press, Cambridge, MA (1988). Sammlung von Originalarbeiten.
- [Bar98] R. Bartak. *Online Guide to Constraint Programming*. <http://kti.ms.mff.cuni.cz/~bartak/constraints> (1998).
- [BB92] K.H. Bläsius und H.-J. Bürcckert. *Deduktionssysteme*. Oldenbourg (1992).
- [BCDS08] A. Billard, S. Calinon, R. Dillmann und S. Schaal. *Robot Programming by Demonstration*. in B. Siciliano und O. Khatib (Hrsg.), *Handbook of Robotics*. Springer (2008), S. 1371–1394.
- [Bel57] R.E. Bellman. *Dynamic Programming*. Princeton University Press (1957).
- [Ben88] H.J. Bentz. *Ein Gehirn für den PC*. c't (1988) 10, 84–96.
- [Ber89] M. Berrondo. *Fallgruben für Kopffüßler*. Fischer Taschenbuch Nr. 8703 (1989).
- [BFOS84] L. Breiman, J. Friedman, R. A. Olshen und C. J. Stone. *Classification and regression trees*. Wadsworth (1984).
- [BHW89] K. Burg, H. Haf und F. Wille. *Höhere Mathematik für Ingenieure, Band 1: Analysis*. Teubner-Verlag, Stuttgart (1989).
- [Bib92] W. Bibel. *Deduktion: Automatisierung der Logik*, Handbuch der Informatik Bd. 6.2. Oldenbourg (1992).
- [Bis05] C.M. Bishop. *Neural networks for pattern recognition*. Oxford University Press (2005).
- [Bis06] C.M. Bishop. *Pattern recognition and machine learning*. Springer New York: (2006).
- [BKI00] C. Beierle und G. Kern-Isberner. *Methoden wissensbasierter Systeme*. Vieweg (2000).
- [BM03] A. G. Barto und S. Mahadevan. *Recent advances in hierarchical reinforcement learning*. Discrete Event Systems, Special issue on reinforcement learning 13 (2003), 41–77.
- [Bra84] V. Braithwaite. *Vehicles – Experiments in Synthetic Psychology*. MIT Press (1984).
- [Bra86] I. Bratko. *PROLOG: Programmierung für Künstliche Intelligenz*. Addison-Wesley (1986).

- [Bra01] B. Brabec. *Computergestützte regionale Lawinenprognose*. Dissertation, ETH Zürich, 2001.
- [Bri91] *Encyclopedia Britannica*. Encyclopedia Britannica Verlag, London (1991).
- [Bur98] C. J. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*. Data Min. Knowl. Discov. **2** (1998) 2, 121–167.
- [CAD] *CADE: conference on automated deduction*. <http://www.cadeconference.org>.
- [Che83] P. Cheeseman. *A Method for Computing Generalised Bayesian Probability Values for Expert Systems*. in Proc. of the 8th Intl. Joint Conf. on Artificial Intelligence (IJCAI-83) (1983).
- [Che85] P. Cheeseman. *In Defense of Probability*. in Proc. of the 9th Intl. Joint Conf. on Artificial Intelligence (IJCAI-85) (1985).
- [CL73] C. L. Chang und R. C. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Orlando, Florida (1973).
- [Cle79] W.S. Cleveland. *Robust Locally Weighted Regression and Smoothing Scatterplots*. Journal of the American Statistical Association **74** (1979) 368, 829–836.
- [CLR90] T. Cormen, Ch. Leiserson und R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Mass (1990).
- [CM94] W.F. Clocksin und C.S. Mellish. *Programming in Prolog*, 4. Aufl. Springer, Berlin, Heidelberg, New York (1994).
- [Coz98] F.G. Cozman. *JavaBayes, Bayesian Networks in Java* (1998). <http://www.cs.cmu.edu/~javabayes>.
- [Das05] J. Dassow. *Logik für Informatiker*. Teubner Verlag (2005).
- [dD91] F.T. de Dombal. *Diagnosis of Acute Abdominal Pain*. Churchill Livingstone (1991).
- [dDLS⁺72] F.T. de Dombal, D.J. Leaper, J.R. Staniland, A.P. McCann und J.C. Horrocks. *Computer aided Diagnosis of acute Abdominal Pain*. British Medical Journal **2** (1972), 9–13.
- [Dee11] *The DeepQA Project*. <http://www.research.ibm.com/deepqa/deepqa.shtml> (2011).
- [DH73] R.O. Duda und P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley (1973). Klassiker zur Bayes-Decision-Theorie.
- [DHS01] R.O. Duda, P.E. Hart und D.G. Stork. *Pattern Classification*. Wiley (2001). Neuauflage des Klassikers [DH73].
- [Dia04] D. Diaz. *GNU PROLOG*. Universität Paris, 2004. Aufl. 1.7, für GNU Prolog version 1.2.18, <http://gnu-prolog.inria.fr>.
- [DNM98] C.L. Blake D.J. Newman, S. Hettich und C.J. Merz. *UCI Repository of machine learning databases*. <http://www.ics.uci.edu/~mlearn/MLRepository.html> (1998).
- [Ede91] E. Eder. *Relative Complexities of First Order Calculi*. Vieweg Verlag (1991).
- [Elk93] C. Elkan. *The Paradoxical Success of Fuzzy Logic*. in Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93). MIT Press (1993), S. 698–703.
- [Ert93] W. Ertel. *Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen*, DISKI Bd. 25. Infix-Verlag, St. Augustin (1993). Dissertation, Technische Universität München.
- [Ert07] W. Ertel. *Künstliche Intelligenz: Homepage zum Buch*. <http://www.hs-weingarten.de/~ertel/kibuch> (2007). Homepage zu diesem Buch mit Arbeitsmaterialien, Demoprogrammen, Links, Literaturverzeichnis, Errata, etc.

- [Ert12] W. Ertel. *Advanced Mathematics for Engineers*. Vorlesungsskript Hochschule Ravensburg-Weingarten: <http://www.hs-weingarten.de/~ertel/vorlesungen/mae/matheng-skript-1213.pdf> (2012).
- [ES99] W. Ertel und M. Schramm. *Combining Data and Knowledge by MaxEnt-Optimization of Probability Distributions*. in *PKDD'99 (3rd European Conference on Principles and Practice of Knowledge Discovery in Databases)*, LNCS Bd. 1704, Springer Verlag, Prague (1999), S. 323–328.
- [ESCT09] W. Ertel, M. Schneider, R. Cubek und M. Tokic. *The Teaching-Box: A Universal Robot Learning Framework*. in *Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009)* (2009). <http://www.servicerobotik.hs-weingarten.de/teachingbox>.
- [ESS89] W. Ertel, J. Schumann und Ch. Suttner. *Learning Heuristics for a Theorem Prover using Back Propagation*. in J. Retti und K. Leidlmair (Hrsg.), 5. Österreichische Artificial-Intelligence-Tagung. Informatik-Fachberichte 208, Springer-Verlag, Berlin, Heidelberg (1989), S. 87–95.
- [FNA⁺09] D. Ferrucci, E. Nyberg, J. Allan, K. Barker, E. Brown, J. Chu-Carroll, A. Ciccolo, P. Duboue, J. Fan und D. Gondek et al. *Towards the open advancement of questionanswer systems*. Technischer Bericht RC24789, IBM, Yorktown Heights, NY (2009). http://www.research.ibm.com/deepqa/question_answerering.shtml.
- [Fra05] Computer Chess Programming Theory. <http://www.frayn.net/beowulf/theory.html> (2005).
- [Fre97] E. Freuder. *In Pursuit of the Holy Grail*. Constraints 2 (1997) 1, 57–61.
- [FS97] B. Fischer und J. Schumann. *SETHEO Goes Software Engineering: Application of ATP to Software Reuse*. in *Conference on Automated Deduction (CADE 97)*. Springer (1997), S. 65–68. <http://ase.arc.nasa.gov/people/schumann/publications/papers/cade97-reuse.html>.
- [GRS03] G. Görz, C.-R. Rollinger und J. Schneeberger (Hrsg.). *Handbuch der Künstlichen Intelligenz*. Oldenbourg Verlag (2003).
- [GT96] M. Greiner und G. Tinhofer. *Stochastik für Studienanfänger der Informatik*. Carl Hanser Verlag, München, Wien (1996).
- [Gue02] G. Guerrerio. *Spektrum der Wissenschaft, Spezial 1/2002: Kurt Gödel*. Spektrum Verlag, Heidelberg (2002).
- [Göd31a] K. Gödel. *Diskussion zur Grundlegung der Mathematik*, Erkenntnis 2. Monatsheft für Mathematik und Physik (1931) 32, 147–148.
- [Göd31b] K. Gödel. *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*. Monatsheft für Mathematik und Physik (1931) 38, 173–198. Englische Version: <http://home.ddc.net/ygg/etext/godel>.
- [HKP91] J. Hertz, A. Krogh und R. Palmer. *Introduction to the theory of neural computation*. Addison Wesley (1991).
- [Hon94] B. Hontschik. *Theorie und Praxis der Appendektomie*. Mabuse Verlag (1994).
- [Hop82] J.J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities*. Proc. Natl. Acad. Sci. USA 79 (April 1982), 2554–2558. Wiederabdruck in [AR88], S. 460–464.

- [HT85] J.J. Hopfield und D.W. Tank. “*Neural*” Computation of Decisions in Optimization Problems. *Biological Cybernetics* (1985) 52, 141–152. Springer.
- [HTF09] T. Hastie, R. Tibshirani und J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 3. Aufl. Springer, Berlin (2009). Online version: <http://www-stat.stanford.edu/~tibs/ElemStatLearn/>.
- [Hüb03] G. Hübner. *Stochastik*. Vieweg Verlag (2003).
- [Jay57] E. T. Jaynes. *Information Theory and Statistical Mechanics*. Physical Review (1957).
- [Jay03] E.T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press (2003).
- [Jen01] F.V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag (2001).
- [Jor99] Michael I. Jordan (Hrsg.). *Learning in graphical models*. MIT Press, Cambridge, MA, USA (1999).
- [Jäh05] B. Jähne. *Digitale Bildverarbeitung*. Springer (2005).
- [Kal01] J.A. Kalman. *Automated Reasoning with OTTER*. Rinton Press (2001). <http://www-unix.mcs.anl.gov/AR/otter/index.html>.
- [Kan89] Th. Kane. Maximum Entropy in Nilsson’s probabilistic Logic. in *Proc. of the 11th Intl. Joint Conf. on Artificial Intelligence (IJCAI-89)* (1989).
- [KK92] J.N. Kapur und H.K. Kesavan. *Entropy Optimization Principles with Applications*. Academic Press (1992).
- [KLM96] L.P. Kaelbling, M.L. Littman und A.P. Moore. *Reinforcement Learning: A Survey*. Journal of Artificial Intelligence Research 4 (1996), 237–285. <http://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a.pdf>.
- [KMK97] H. Kimura, K. Miyazaki und S. Kobayashi. Reinforcement Learning in POMDPs with Function Approximation. in *14th International Conference on Machine Learning*. Morgan Kaufmann Publishers (1997), S. 152–160. <http://sysplan.nams.kyushu-u.ac.jp/gen/papers/JavaDemoML97/robodemo.html>.
- [Koh72] T. Kohonen. Correlation matrix memories. IEEE Transactions on Computers (1972) C-21, 353–359. Wiederabdruck in [AR88], S. 171–174.
- [Kre06] Ch. Kreitz. *Formale Methoden der Künstlichen Intelligenz*. Künstliche Intelligenz (2006) 4, 22–28.
- [Lar00] F.D. Laramée. *Chess Programming, Part 1–6*. <http://www.gamedev.net/reference/programming/features/chess1> (2000).
- [Le999] *LEXMED – lernfähiges Expertensystem für medizinische Diagnose*. <http://www.lexmed.de> (1999).
- [Let03] R. Letz. *Praktikum Beweiser*. <http://www4.in.tum.de/~letz/PRAKTIKUM/al-ss05.pdf> (2003).
- [Lif89] V. Lifschitz. Benchmark Problems for formal Non-Monotonic Reasoning. in Reinfrank et al (Hrsg.), *Non-Monotonic Reasoning: 2nd International Workshop*, LNAI Bd. 346. Springer (1989), S. 202–219.
- [LSBB92] R. Letz, J. Schumann, S. Bayerl und W. Bibel. *SETHEO: A High-Performance Theorem Prover*. Journal of Automated Reasoning (1992) 8, 183–212. <http://www4.informatik.tu-muenchen.de/~letz/setheo>.
- [McC] W. McCune. *Automated Deduction Systems and Groups*. <http://www-unix.mcs.anl.gov/AR/others.html>. siehe auch <http://www-formal.stanford.edu/clt/ARS/systems.html>.

- [McD82] J. McDermott. *R1: A Rule-Based Configurer of Computer Systems*. Artificial Intelligence **19** (1982), 39–88.
- [MDBM00] G. Melancon, I. Dutour und G. Bousque-Melou. *Random Generation of Dags for Graph Drawing*. Technischer Bericht INS-R0005, Dutch Research Center for Mathematical and Computer Science (CWI) (2000). <http://ftp.cwi.nl/CWIreports/INS/INS-R0005.pdf>.
- [Mit97] T. Mitchell. *Machine Learning*. McGraw Hill (1997). <http://www-2.cs.cmu.edu/~tom/mlbook.html>.
- [MP69] M. Minsky und S. Papert. *Perceptrons*. MIT Press, Cambridge, MA (1969).
- [Nil86] N. J. Nilsson. *Probabilistic Logic*. Artificial Intelligence (1986) 28, 71–87.
- [Nil98] N. Nilsson. *Artificial Intelligence – A New Synthesis*. Morgan Kaufmann (1998).
- [NPW02] T. Nipkow, L.C. Paulson und M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, LNCS Bd. 2283. Springer (2002). <http://www.cl.cam.ac.uk/Research/HVG/Isabelle>.
- [NS61] A. Newell und H. A. Simon. *GPS, A Program that Simulates Human Thought*. in H. Billing (Hrsg.), *Lernende Automaten*. Oldenbourg, München (1961), S. 109–124.
- [NSS83] A. Newell, J. C. Shaw und H. A. Simon. *Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics*. in J. Siekmann und G. Wrightson (Hrsg.), *Automation of Reasoning 1: Classical Papers on Computational Logic 1957–1966*. Springer, Berlin, Heidelberg (1983), S. 49–73. Erstpublikation: 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*. Der Chirurg **66** (1995), 135–141.
- [OMYL96] C. Ohmann, V. Moustakis, Q. Yang und K. Lang. *Evaluation of automatic knowledge acquisition techniques in the diagnosis of acute abdominal pain*. Art. Intelligence in Medicine **8** (1996), 23–36.
- [OPB94] C. Ohmann, C. Platen und G. Belenky. *Computerunterstützte Diagnose bei akuten Bauchschmerzen*. Chirurg **63** (1994), 113–123.
- [Pal80] G. Palm. *On Associative Memory*. Biological Cybernetics **36** (1980), 19–31.
- [Pal91] G. Palm. *Memory capacities of local rules for synaptic modification*. Concepts in Neuroscience **2** (1991) 1, 97–128. MPI Tübingen.
- [PB06] T. Pellegrini und A. Blumauer. *Semantic Web: Wege zur vernetzten Wissensgesellschaft*. Springer (2006).
- [Pea84] J. Pearl. *Heuristics, Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company (1984).
- [Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference*. Morgan Kaufmann (1988).
- [PL05] L. Panait und S. Luke. *Cooperative Multi-Agent Learning: The State of the Art*. Autonomous Agents and Multi-Agent Systems **11** (2005) 3, 387–434.
- [Pól95] George Pólya. *Schule des Denkens — Vom Lösen mathematischer Probleme*. Francke Verlag (1995).
- [PS08] J. Peters und S. Schaal. *Reinforcement learning of motor skills with policy gradients*. Neural Networks **21** (2008) 4, 682–697.

- [Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers (1993). C4.5 Download: <http://www.rulequest.com/Personal>, C5.0 Bestellung: <http://www.rulequest.com>.
- [Rau96] W. Rautenberg. *Einführung in die Mathematische Logik*. Vieweg Verlag (1996).
- [RB93] M. Riedmiller und H. Braun. *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*. in *Proceedings of the IEEE International Conference on Neural Networks* (1993), S. 586–591.
- [RGH⁺06] M. Riedmiller, T. Gabel, R. Hafner, S. Lange und M. Lauer. *Die Brainstormers: Entwurfsprinzipien lernfähiger autonomer Roboter*. Informatik-Spektrum **29** (2006) 3, 175 – 190.
- [RHR86] D.E. Rumelhart, G.E. Hinton und Williams R.J. *Learning Internal Representations by Error Propagation*. in [RM86] (1986).
- [Ric83] E. Rich. *Artificial Intelligence*. McGraw-Hill (1983).
- [Ric03] M. Richter. *Fallbasiertes Schließen*. in [GRS03], Kap. 11, S. 407–430.
- [RM86] D. Rumelhart und J. McClelland. *Parallel Distributed Processing*, Bd. 1. MIT Press (1986).
- [RM96] W. Rödder und C.-H. Meyer. *Coherent Knowledge Processing at Maximum Entropy by SPIRIT*. in *KI-96 (German national conference on AI)*, Dresden (1996).
- [RMD07] M. Riedmiller, M. Montemerlo und H. Dahlkamp. *Learning to Drive a Real Car in 20 Minutes*. in *FBIT '07: Proceedings of the 2007 Frontiers in the Convergence of Bioscience and Information Technologies*, IEEE Computer Society, Washington, DC, USA (2007), S. 645–650.
- [RMS91] H. Ritter, T. Martinez und K. Schulten. *Neuronale Netze*. Addison Wesley (1991).
- [RN03] S. Russell und P. Norvig. *Artificial Intelligence: A Modern Approach*, 2. Aufl. Prentice Hall (2003). 1. Auflage 1995, deutsche Übersetzung der 2. Auflage 2004 bei Pearson Studium, <http://aima.cs.berkeley.edu>.
- [Roba] *RoboCup Official Site*. <http://www.robocup.org>.
- [Robb] *The RoboCup Soccer Simulator*. <http://sserver.sourceforge.net>.
- [Rob65] J.A. Robinson. *A machine-oriented logic based on the resolution principle*. Journal of the ACM **12** (1965) 1, 23–41.
- [Roj93] R. Rojas. *Theorie der neuronalen Netze*. Springer (1993).
- [Ros58] F. Rosenblatt. *The perceptron : a probabilistic model for information storage and organization in the brain*. Psychological Reviews **65** (1958), 386–408. Wiederabdruck in [AR88], S. 92–114.
- [RW06] C.E. Rasmussen und C.K.I. Williams. *Gaussian Processes for Machine Learning*. Mit Press (2006). Online version: <http://www.gaussianprocess.org/gpml/chapters/>.
- [SA94] S. Schaal und C.G. Atkeson. *Robot juggling: implementation of memory-based learning*. IEEE Control Systems Magazine **14** (1994) 1, 57–71.
- [Sam59] A.L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal **1** (1959) 3, 210–229.
- [SB98] R. Sutton und A. Barto. *Reinforcement Learning*. MIT Press (1998). <http://www.cs.ualberta.ca/~sutton/book/the-book.html>.
- [SB04] J. Siekmann und Ch. Benzmüller. *Omega: Computer Supported Mathematics*. in *KI 2004: Advances in Artificial Intelligence*, LNAI 3238. Springer Verlag (2004), S. 3–28. <http://www.ags.uni-sb.de/~omega>.

- [Sch96] M. Schramm. *Indifferenz, Unabhängigkeit und maximale Entropie: Eine wahrscheinlichkeitstheoretische Semantik für Nicht-Monotones Schließen*. Dissertationen zur Informatik Nr. 4. CS-Press, München (1996).
- [Sch01] J. Schumann. *Automated Theorem Proving in Software Engineering*. Springer Verlag (2001).
- [Sch02] S. Schulz. *E – A Brainiac Theorem Prover*. Journal of AI Communications **15** (2002) 2/3, 111–126. <http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html>.
- [Sch04] A. Schwartz. *SpamAssassin*. O'Reilly (2004). Spamassassin-Homepage: <http://spamassassin.apache.org>.
- [SE90] Ch. Suttner und W. Ertel. *Automatic Acquisition of Search Guiding Heuristics*. in *10th Int. Conf. on Automated Deduction*. Springer-Verlag, LNAI 449 (1990), S. 470–484.
- [SE00] M. Schramm und W. Ertel. *Reasoning with Probabilities and Maximum Entropy: The System PIT and its Application in LEXMED*. in K. Inderfurth et al (Hrsg.), *Operations Research Proceedings (SOR'99)*. Springer Verlag (2000), S. 274–280.
- [SE10] M. Schneider und W. Ertel. *Robot Learning by Demonstration with Local Gaussian Process Regression*. in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10)* (2010).
- [Sho76] E.H. Shortliffe. *Computer-based medical consultations, MYCIN*. North-Holland, New York (1976).
- [Spe03] *Spektrum der Wissenschaft, Spezial 5/2003: Intelligenz*. Spektrum Verlag, Heidelberg (2003).
- [Spe04] *Spektrum der Wissenschaft, Spezial 1/2004: Bewusstsein*. Spektrum Verlag, Heidelberg (2004).
- [SR86] T.J. Sejnowski und C.R. Rosenberg. *NETtalk: a parallel network that learns to read aloud*. Technischer Bericht JHU/EECS-86/01, The John Hopkins University Electrical Engineering and Computer Science Technical Report (1986). Wiederabdruck in [AR88], S. 661–672.
- [SS02] S. Schölkopf und A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press (2002).
- [SS06] G. Sutcliffe und C. Suttner. *The State of CASC*. AI Communications **19** (2006) 1, 35–48. CASC-Homepage: <http://www.cs.miami.edu/~tptp/CASC>.
- [SSK05] P. Stone, R.S. Sutton und G. Kuhlmann. *Reinforcement Learning for RoboCup-Soccer Keepaway*. Adaptive Behavior (2005). To appear. <http://www.cs.utexas.edu/~pstone/Papers/bib2html-links/AB05.pdf>.
- [SW76] C.E. Shannon und W. Weaver. *Mathematische Grundlagen der Informationstheorie*. Oldenbourg Verlag (1976).
- [Sze10] C. Szepesvari. *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers (2010). draft available online: <http://www.ualberta.ca/~szepesva/RLBook.html>.
- [Tax01] D.M.J. Tax. *One-class classification*. Dissertation, Delft University of Technology, 2001.
- [Ted08] R. Tedrake. *Learning Control at Intermediate Reynolds Numbers*. in *Workshop on: Robotics Challenges for Machine Learning II, International Conference on Intelligent Robots and Systems (IROS 2008)*, Nizza, Frankreich (2008).

- [TEF09] M. Tokic, W. Ertel und J. Fessler. *The Crawler, A Class Room Demonstrator for Reinforcement Learning (to appear)*. in In Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference (FLAIRS 09), AAAI Press, Menlo Park, California (2009).
- [Tes95] G. Tesauro. *Temporal Difference Learning and TD-Gammon*. Communications of the ACM **38** (1995) 3. <http://www.research.ibm.com/massive/tdl.html>.
- [Tok06] M. Tokic. *Entwicklung eines Lernfähigen Laufroboters*. Diplomarbeit Hochschule Ravensburg-Weingarten (2006). Inklusive Simulationssoftware verfügbar auf <http://www.hs-weingarten.de/~ertel/kibuch>.
- [Tur37] A.M. Turing. *On computable numbers, with an application to the Entscheidungsproblem*. Proceedings of the London Mathemat. Society **42** (1937) 2.
- [Tur50] A.M. Turing. *Computing Machinery and Intelligence*. Mind **59** (1950), 433–460. Deutsche Übersetzung mit dem Titel *Kann eine Maschine denken* in [ZW94].
- [vA06] L. v. Ahn. *Games With a Purpose*. IEEE Computer Magazine (Juni 2006), 96–98. <http://www.cs.cmu.edu/~biglou/ieee-gwap.pdf>.
- [Wei66] J. Weizenbaum. *ELIZA-A Computer Program For the Study of Natural Language Communication Between Man and Machine*. Communications of the ACM **9** (1966) 1, 36–45.
- [WF01] I. Witten und E. Frank. *Data Mining*. Hanser Verlag München (2001). Von den Autoren in Java entwickelte DataMining Programmhbibliothek WEKA: (<http://www.cs.waikato.ac.nz/~ml/weka>).
- [Whi96] J. Whittaker. *Graphical models in applied multivariate statistics*. Wiley (1996).
- [Wie] U. Wiedemann. *PhilLex, Lexikon der Philosophie*. <http://www.phillex.de/paradoxa.htm>.
- [Wie04] J. Wielemaker. *SWI-Prolog 5.4*. Universität Amsterdam, 2004. <http://www.swi-prolog.org>.
- [Wik13] *Wikipedia, die freie Enzyklopädie*. <http://de.wikipedia.org> (2013).
- [Win] P. Winston. *Game Demonstration*. <http://www.ai.mit.edu/courses/6.034f/gamepair.html>. Java Applet zu Minimax- und Alpha-Beta-Suche.
- [Zdz05] J. Zdziarski. *Ending Spam*. No Starch Press (2005).
- [Zel94] A. Zell. *Simulation Neuronaler Netze*. Addison Wesley (1994). Im Buch beschriebener Simulator SNNS, bzw. JNNS: <http://www-ra.informatik.uni-tuebingen.de/SNNS>.
- [ZSR⁺99] A. Zielke, H. Sitter, T.A. Rampp, E. Schäfer, C. Hasse, W. Lorenz und M. Rothmund. *Überprüfung eines diagnostischen Scoresystems (Ohmann-Score) für die akute Appendizitis*. Chirurg **70** (1999), 777–783.
- [ZW94] W.C. Zimmerli und S. Wolf (Hrsg.). *Künstliche Intelligenz – Philosophische Probleme*. Philipp Reclam, Stuttgart (1994).

Sachverzeichnis

A

Abhängigkeitsgraph, 150
Ableitung, 24
Abstandsmaß, 231
Adaptive Resonance Theory, 284
Agent, 3, 12, 287, 290, 291, 295, 296, 298, 299, 301, 303, 304
autonomer, 9
Hardware-, 12
intelligenter, 12
kostenorientierter, 13, 153
lernender, 9, 180
lernfähiger, 14
mit Gedächtnis, 12
nutzenorientierter, 14
Reflex-, 12
Software-, 12
zielorientierter, 12
Agenten, verteilte, 9, 14
Ähnlichkeit, 193
Aktion, 97, 290
Aktionen, 97
Aktivierungsfunktion, 250, 261
Aktuator, 12
Alarm-Beispiel, 159
 A^* -Algorithmus, 109, 278
allgemeingültig, 21
Alpha-Beta-Pruning, 115, 118
Antinomien, 68
Appendizitis, 133, 144
Approximation, 180, 197
 lineare, 273
 modellfreie, 197
A-priori-Wahrscheinlichkeit, 134
a-priori-Wahrscheinlichkeit, 131
Äquivalenz, 20

ART

, 284
Assoziationsfehler, 264
Assoziativspeicher, 260
Attribut, 119, 204
Aussage, 39
Aussagenlogik, 15, 19
Aussagevariablen, 19
Autoassoziativspeicher, 253, 260, 266

B

Backgammon, 120, 304
Backpropagation, 264, 274, 279, 296, 306
 -Lernregel, 275
Backtracking, 76, 102
backward chaining, 32
Batch-Learning, 223
Batch-Lernverfahren, 271
Bayes
 -Formel, 134, 151, 163, 170, 223
 -Netz, 9, 11, 72, 127, 158, 160, 202
 -Netz, lernendes, 172, 220
BDD, 34
bedingt unabhängig, 160, 161, 170
Belegung, 20, 40
Bellman
 -Gleichung, 294
 -Prinzip, 293, 294
Belohnung
 abgeschwächte, 290
 direkte, 290, 299
beobachtbar, 99, 114
beschränkte Ressourcen, 105
Beweisverfahren, 24
Bewertungsfunktion, 115
bias unit, 190

Bias-Variance-Tradeoff, 219
 binäre Entscheidungsdiagramme, 34
 Boltzmann-Maschine, 259
 Braatenberg-Vehikel, 2, 9, 196
 Built-in-Prädikat, 84

C

C4.5, 203, 221
 CART, 203, 217
 CASC, 55
 case-based reasoning, 201
 CBR, 201
 Certainty Factors, 126
 chatterbot, 4
 Church, Alonso, 6
 CLP, 86
 Cluster, 230
 Clustering, 230, 240
 hierarchisches, 233

cognitive science, 4

Computerdiagnose, 158
 conditional probability table, 160
 confusion matrix, 13
 constraint, 86
 Constraint Logic Programming, 86
 Constraint Satisfaction Problem, 86
 CPT, 160, 163, 171, 220, 223
 credit assignment, 120, 290
 CSP, 86
 curse of dimensionality, 305
 Cut, 79

D

DAG, 170, 221
 DAI, 9
 Dame, 114, 120, 121
 Data Mining, 9, 181, 182, 201, 203, 215
 Default-Logik, 71
 Default-Regel, 71
 Delta-Regel, 271–273
 verallgemeinerte, 275
 Demodulation, 54
 deMorgan, 44
 Dempster-Schäfer-Theorie, 127
 deterministisch, 99, 114
 Diagnosesystem, 145
 Disjunktion, 20, 26

distributed artificial intelligence, 9
 d-Separation, 171
 Dynamische Programmierung, 293

E

eager learning, 199
 E-Learning, 6
 Elementarereignis, 128
 Eliza, 4
 EM-Algorithmus, 223, 233
 Entropie, 206
 maximale, 127, 135
 Entscheidung, 153
 Entscheidungsbaum, 15, 204
 Induktion, 181
 Entscheidungsbaumlernen, 202, 306
 Ereignis, 128
 erfüllbar, 21
 Expertensystem, 144, 158

F

Fakt, 31
 Faktorisierung, 28, 52
 Fallbasiertes Schließen, 201
 Fallbasis, 201
 falsch negativ, 154
 falsch positiv, 154
 Farthest Neighbour-Algorithmus, 235
 Feature, 119
 Finite-Domain-Constraint-Solver, 88
 forward chaining, 32
 Frame-Problem, 71
 freie Variablen, 39
 Funktionsapproximation, 239
 Funktionssymbol, 38
 Fuzzy-Logik, 9, 15, 34, 72, 127

G

Gauß'sche Prozesse, 198, 239
 General Problem Solver, 10
 Generalisierung, 178
 Genetic Programming, 83
 geschlossene Formel, 39
 gierige Suche, 107, 108, 222
 Gleichungen
 gerichtete, 54

- Go, 114, 120, 122
goal stack, 33
Gödel
 Kurt, 6
 scher Unvollständigkeitssatz, 6, 68
 scher Vollständigkeitssatz, 6, 48
GPS, 10
Gradientenabstieg, 273
greedy search, 107, 108
Grundterm, 48
- H**
halbentscheidbar
 PL1, 67
Halteproblem, 6
Hebb-Regel, 252, 261, 276
 binäre, 264
Heuristik, 53, 66, 105
heuristische Bewertungsfunktion, 106, 109
hierarchisches Lernen, 306
Hirnforschung, 3
Hopfield-Netz, 253, 254, 265
Hornklausel, 31, 80, 81
Hugin, 164
- I**
ID3, 203
IDA^{*}-Algorithmus, 111, 278
Implikation, 20
Impuls, 280
indifferent, 139
Induktion von Entscheidungsbäumen, 203
induktive Statistik, 72
Inferenzmaschine, 49
Inferenzmechanismus, 14
Informationsgehalt, 207
Informationsgewinn, 205, 208, 239
inkrementeller Gradientenabstieg, 273
inkrementelles Lernen, 271
Input-Resolution, 53
Interpretation, 20, 40
Iterative Deepening, 103, 111, 118
- J**
JavaBayes, 164
- K**
Kalkül, 25
 Gentzen, 47
 natürliches Schließen, 47
 Sequenzen, 47
Kernel, 198, 282
Kernel-Methode, 282
Kettenregel für Bayes-Netze, 133, 169, 170
KI, 1
Klassifikation, 178
Klassifizierer, 178, 179, 228, 273
Klausel
 definite, 31
 -Kopf, 31
k-Means, 232
k-Nearest-Neighbour-Methode, 195, 197, 198, 218, 243
KNF, 26
KNIME, 203, 236
Knoten, 106
Knowledge Engineering, 14
Kognitionswissenschaft, 3, 4
Kohonen, 261
komplementär, 27
Konditionierung, 163, 170
Konfusionsmatrix, 236
Konjunktion, 20, 26
konjunktive Normalform, 26
Konklusion, 31
Konnektionismus, 8
konsistent, 28, 138
Konstanten, 38
korrekt, 25
Korrelation, 150
Korrelationskoeffizienten, 184
Korrelationsmatrix, 241
Kostenfunktion, 97, 108
Kostenmatrix, 149, 153
Kostenschätzfunktion, 107
Kreuzvalidierung, 199, 217, 218
Künstliche Intelligenz, 1
- L**
Laplace-Annahme, 129
Laplace-Wahrscheinlichkeit, 129
lazy learning, 199
Leave-One-Out-Kreuzvalidierung, 219, 243
Lernen, 178

- auswendig, 178
- Batch-, 271
- durch Demonstration, 306, 307
- durch Verstärkung, 100, 120, 239, 287
- eifriges, 199
- faules, 199
- hierarchisches, 306
- inkrementell, 223, 271
- maschinelles, 150, 177
- mit einer Klasse, 228
- mit Lehrer, 177, 230, 266
- Multi-Agenten-, 306
- TD, 302, 304
- verteiltes, 306
- lernender Agent, 180
- Lernphase, 180
- Lernrate, 252, 272
- LEXMED, 127, 135, 144, 211
- linear separabel, 185, 186
- LIPS, 76
- LISP, 7, 10
- Literal, 26
 - komplementäres, 27
- locally weighted linear regression, 201
- Logic Theorist, 7, 10
- Logik
 - höherer Stufe, 68, 69
 - probabilistische, 15, 34
- Lösung, 97

- M**
- Manhattan-Abstand, 112, 231
- Marginalisierung, 133–135
- Markov decision process, 291
 - partially observable, 291
- Markov-Entscheidungsprozess, 12, 291, 304
 - nichtdeterministischer, 301
- maschinelles Lernen, 147
- materiale Implikation, 20, 127, 141
- MaxEnt, 127, 139, 141, 144, 147, 163, 171
- MaxEnt-Verteilung, 138
- MDP, 291, 293, 304
 - deterministischer, 299
 - nichtdeterministischer, 301
- medizinische Diagnose, 144
- memory-based learning, 200, 201
- Merkmal, 178, 187, 204
- Merkmalsraum, 179

- Metasprache, 21
- Methode der kleinsten Quadrate, 157, 239, 269–271, 273
- MGU, 51
- minimal aufspannender Baum, 234
- Modell, 21
- Modellkomplexität, 218, 219
- Modus Ponens, 26, 35, 126, 138
- monoton, 69, 143
- Monotonie, 69
- Multi-Agenten-Lernen, 306
- Multiagentensysteme, 11
- MYCIN, 10, 126, 144

- N**
- Nachbedingung, 59
- Naive Reverse, 82
- Naive-Bayes, 156, 158, 172, 182, 192, 224–226, 245, 331
- Naive-Bayes-Klassifizierer, 223, 225, 226
- Nearest Neighbour Data Description, 228
- Nearest Neighbour-Algorithmus, 234
- Nearest Neighbour-Klassifikation, 193
- Nearest Neighbour-Methode, 192, 228
- Negation, 20
 - Negation as Failure, 80
- Neuroinformatik, 259
- Neuronale Netze, 198
- neuronale Netze, 6, 10, 197, 247
 - rekurrente, 253, 259
- Neurotransmitter, 248
- nichtmonotone Logik, 71
- Normalform
 - konjunktive, 26
 - pränexe, 44
- Normalgleichungen, 270
- Nullsummenspiel, 114

- O**
- Ockhams Rasiermesser, 215, 216
- Oder-Verzweigungen, 79
- Offline-Algorithmen, 100
- One-Class-Learning, 228
- Online-Algorithmen, 100
- Ontologie, 61
- orthonormal, 262
- Outlier Detection, 228

overfitting, 195, 216, 222, 268, 281, 304
OWL, 61

P

Paramodulation, 54
Perzeptron, 10, 185, 186, 251
Phasenübergang, 257
Pinguin-Problem, 86
PIT, 142, 143, 163, 172
PL1, 15, 38
Planen, 84
Policy-Gradient-Methode, 304
POMDP, 291, 304
Prädikatenlogik, 6
 erster Stufe, 15, 38
Prämisse, 31
probabilistische Logik, 15
probabilistisches Schließen, 9
Produktregel, 132
Programmverifikation, 59
Prolog, 7, 10, 24, 33, 75
Pruning, 211, 217
Pseudoinverse, 263
Pure Literal-Regel, 53, 63

Q

Q-Lernen, 298, 306
 Konvergenz, 299
Quickprop, 280

R

Randverteilung, 133
Rapid Prototyping, 88
Rauschen, 195
RDF, 61
Realzeitanforderungen, 115
Realzeitentscheidung, 105
Receiver Operating Characteristic, 155
reinforcement learning, 120, 287
Resolution, 27
 SLD-, 32
Resolutionskalkül, 10, 24
Resolutionsregel, 27, 35
 allgemeine, 27, 50
Resolvente, 27
Reversi, 114

reward, immediate, 290
Risikomanagement, 154
RoboCup, 11, 304
Roboter, 12
ROC-Kurve, 155, 156, 237
RProp, 280

S

Schach, 94, 114, 117–122
Score, 144, 156, 224, 226, 245, 270
selbstorganisierende Karten, 284
Semantic Web, 61
Semantik
 deklarative (Prolog), 78
 prozedurale (Prolog), 78, 81
Semantische Bäume, 33
Semi-Supervised Learning, 239
Sensitivität, 155, 162
Sensor, 12
Set of Support-Strategie, 53, 63
Sigmoid-Funktion, 252, 269, 274
Signatur, 19
simulated annealing, 259
Situationskalkül, 71
Skolemisierung, 46
SLD-Resolution, 35
Software Wiederverwendung, 59
Spam, 225
Spam-Filter, 13, 225, 227
Sparsamkeitsprinzip, 215
Spezifität, 155
Startzustand, 97
Statistische Induktion, 150
Strategie, 290
Strategie, optimale, 291
Streudiagramm, 179
Subgoal, 77
Substitutionsaxiom, 43
Subsumption, 53
Suchalgorithmus, 96
 optimaler, 99
 vollständiger, 98
Suchbaum, 97
Suche
 heuristische, 94
 uninformierte, 94
Suchraum, 28, 32, 47, 53
Support Vector Data Description, 228

Support Vector Machine, 239
 Support Vektor Maschine, 198
 Support-Vektor, 281
 Support-Vektor-Maschine, 281, 303
 SVM, 281

T

Tautologie, 21
 TD-Fehler, 302
 TD-Gammon, 304
 TD-learning, 302, 304
 TD-Lernen, 302, 304
 Teaching-Box, 307
 Teilziel, 32
 temporal difference error, 302
 temporal difference learning, 302
 Term, 38
 Termersetzungssystem, 54
 Testdaten, 181, 215
 Text Mining, 182
 Textklassifikation, 225
 Theorembeweiser, 7, 49, 54, 59, 60
 Tiefenschränke, 103
 Trainingsdaten, 67, 180, 215
 Trainingsdatum, 204
 Turing, Alan, 6
 Turing-Test, 4
 Tweety-Beispiel, 69, 71, 143

U

Überanpassung, 195, 216, 218–220, 222, 268, 271, 304
 Übergangsfunktion, 290, 301
 Umgebung, 12, 14
 beobachtbare, 14
 deterministische, 14
 diskrete, 14
 nichtdeterministische, 14
 stetige, 14
 teilweise beobachtbare, 14
 unabhängig, 132
 bedingt, 160, 161, 170
 Und-Oder-Baum, 79
 Und-Verzweigungen, 79
 unerfüllbar, 21
 Unifikation, 50
 Unifikator, 51

allgemeinster, 51
 unifizierbar, 51
 Uniform Cost Search, 101
 Unit-Klausel, 53
 Unit-Resolution, 53

V

value iteration, 294
 Variablen, 38
 VDML-SL, 60
 Verstärkung
 Lernen durch, 290
 negative, 290
 positive, 290
 verteiltes Lernen, 306
 Verteilung, 130, 146
 Verzweigungsfaktor, 93, 97
 effektiver, 98
 mittlerer, 95
 Vienna Development Method Specification Language, 60
 vollständig, 25
 Vorbedingung, 59
 Voronoi-Diagramm, 194

W

wahr, 21, 41
 Wahrheitsmatrix, 13
 Wahrheitstabelle, 21
 Wahrheitstafelmethode, 24
 Wahrscheinlichkeit, 127, 129
 bedingte, 131
 Wahrscheinlichkeitslogik, 71
 Wahrscheinlichkeitsregeln, 150
 Wahrscheinlichkeitsverteilung, 130
 WAM, 76, 83
 Warren-abstract-machine, 76
 WB, 14
 WEKA, 203, 236
 Welt, 20
 Wert, 290
 Wert-Iteration, 294
 widerspruchsfrei, 28
 Wissen, 14
 Wissensbasis, 14, 23, 159
 widerspruchsfreie, 28
 Wissens-Ingenieur, 11

- Wissensingenieur, 14
Wissensquelle, 14
- Z**
- Ziel, 32
- Zielfunktion, 179
Zielzustand, 97
Zufallsvariable, 128
zulässig, 109
Zustand, 97, 106, 289, 290
Zustandsraum, 97