Einführung in die Künstliche Intelligenz

G. Görz, C. Rollinger, J. Schneeberger (Hrsg.)

15. Juni 2000

Inhaltsverzeichnis

Kapitel 1

Maschinelles Lernen und Data Mining

Katharina Morik, Stefan Wrobel und Thorsten Joachims

Das folgende Kapitel gibt eine Einführung in die wichtigsten Themen, theoretischen Grundlagen, Verfahren und Anwendungen des Forschungsgebietes "maschinelles Lernen". Das maschinelle Lernen (vom englischen *machine learning*) beschäftigt sich mit der *computergestützten Modellierung und Realisierung von Lernphänomenen.* Durch diese Definition ist der Gegenstandsbereich des Gebietes zwar umfassend, aber nicht präzise beschrieben. Der nächste Abschnitt wird sich daher zunächst der Frage widmen, wie Lernen und maschinelles Lernen angemessen definiert werden können, und einen Überblick geben über die Entwicklung und den aktuellen Zustand dieses Forschungs- und Anwendungsgebietes inklusive der Bereiche Data Mining bzw. Wissensentdeckung.

Im Abschnitt 1.2 führen wir danach die Lernaufgabe ein, mit der sich ein großer Teil des maschinellen Lernens und auch dieses Kapitels beschäftigen: das Lernen von Funktionen aus Beispielen. Die darauffolgenden Abschnitte beschreiben verschiedene Spezialisierungen dieser Aufgabe zusammen mit prototypischen Verfahren zu ihrer Lösung: Entscheidungsbäume (Abschnitt 1.3), instanzbasiertes Lernen (Abschnitt 1.4), Stützvektormethode (Abschnitt 1.5). Da den neuronalen Netzen in diesem Buch ein eigenes Kapitel gewidmet ist, werden sie hier nicht noch einmal behandelt. Auch genetische Programmierung oder evolutionäre Strategien fehlen, obwohl sie durchaus zum Funktionslernen eingesetzt werden können. Es folgt ein Abschnitt, der sich mit der grundlegenden theoretischen Lernbarkeit von Funktionen aus der Sicht des sog. PAC-Lernens (Abschnitt 1.6) beschäftigt (die Sicht der Identifikation im Grenzwert [Angluin und Smith, 1983], [Lange und Zeugmann, 1992] wird hier nicht dargestellt). Da sich aus der Verwendung prädikatenlogischer Hypothesenräume besondere Herausforderungen ergeben, widmen sich die folgenden Abschnitte speziell diesem Teilbereich des maschinellen Lernens. Wir beginnen mit dem Funktionslernen (Begriffslernen) (Abschnitt 1.7) und fahren in Abschnitt 1.8 mit der Aufgabe der Revision logischer Theorien fort. Auch für die logischen Verfahren folgt ein Abschnitt (1.9), der sich mit theoretischen Lernbarkeitsfragen befasst.

In weiteren Abschnitten wenden wir uns danach verschiedenen *deskriptiven* Lernauf-

gaben zu. In Abschnitt 1.10 erörtern wir Verfahren zum assoziativen Regellernen, gefolgt von der Aufgabe der Subgruppenentdeckung in Abschnitt 1.11 und der Clusteranalyse in Abschnitt 1.12. Schließlich diskutieren wir Verfahren zum Erlernen optimaler Handlungsvorschriften aus Erfahrungen, das so genannte Verstärkungslernen (Abschnitt 1.13).

1.1 Was ist maschinelles Lernen

Wie oben ausgeführt bezeichnet man als "maschinelles Lernen" ein Forschungsgebiet, das sich mit der *computergestützten Modellierung und Realisierung von Lernphänomenen* beschäftigt. Dabei ist es durchaus nützlich, sich bei der Interpretation des für das Gebiet zentralen Schlüsselbegriffs "Lernen" zunächst von der intuitiven Bedeutung dieses Wortes leiten zu lassen, denn es ist, wie wir nun sehen werden, nicht einfach, eine zugleich präzise und umfassende intensionale Definition von "Lernen" zu geben.

1.1.1 Intensionale Definitionsversuche

Es ist nicht einfach, unser intuitives Verständnis von Lernen in eine präzise und umfassende Definition zu fassen, wie zwei klassische Definitionen von "Lernen" belegen. So definiert Herbert Simon [Simon, 1983]:

> Lernen ist jeder Vorgang, der ein System in die Lage versetzt, bei der zukünftigen Bearbeitung derselben oder einer ähnlichen Aufgabe diese besser zu erledigen.

Diese Definition wurde unter anderem von Ryszard Michalski, einem weiteren Pionier des maschinellen Lernens, kritisiert, weil das Ziel beim Lernen nicht immer offensichtlich ist, und es auch Lernen ohne konkretes vorgefasstes Ziel geben kann, z. B. wenn man durch eine Stadt läuft und sich merkt, wo sich die Bibliothek befindet, ohne konkret vorzuhaben, die Bibliothek aufzusuchen. Auch ist nicht jede verbesserte Aufgabenlösung Lernen (ein geschärftes Messer schneidet besser, hat aber nicht gelernt). Als alternative Definition hat Michalski daher vorgeschlagen [Michalski, 1986]:

> Lernen ist das Konstruieren oder Verändern von Repräsentationen von Erfahrungen.

Diese Definition ist einerseits bereits so allgemein, daß sie kaum noch nützlich ist, und kann andererseits immer noch als zu eng kritisiert werden, denn nicht immer müssen beim Lernen Repräsentationen im engeren Sinne im Spiel sein.

Wesentlich präziser, allerdings deutlich weniger umfassend, ist demgegenüber die Zurückführung auf eine der drei klassischen logischen Schlußformen, nämlich den induktiven Schluß. Die drei Schlußformen werden gerne anhand der Sterblichkeit von Sokrates illustriert:

> **Deduktion:** Aus "Alle Menschen sind sterblich." und "Sokrates ist ein Mensch." schließe "Sokrates ist sterblich." (wahrheitserhaltend).
>
> **Induktion:** Aus "Sokrates ist ein Mensch." und "Sokrates ist sterblich." schließe "Alle Menschen sind sterblich." (falschheitserhaltend bzgl. der zweiten Aussage).

> **Abduktion:** Aus "Sokrates ist sterblich." und "Alle Menschen sind sterblich." schließe "Sokrates ist ein Mensch." (falschheitserh. bzgl. der ersten Aussage).

Während wohl die meisten Forscher zustimmen würden, dass der induktive Schluß vom Besonderen auf das Allgemeine ein wesentliches Element des maschinellen Lernens ist, so schließt auch diese Definition viele Spielarten des Lernens aus und ist nicht so präzise, daß mit ihrer Hilfe genau entschieden werden könnte, ob ein bestimmtes Phänomen ein Lernphänomen ist.

1.1.2 Extensionale Definition über Lernaufgaben

Alternativ kann man deshalb versuchen, das maschinelle Lernen extensional über die einzelnen Typen von *Lernaufgaben* zu definieren, mit denen man sich (bisher) in diesem Gebiet beschäftigt. Dies hat den Vorteil, dass eine einzelne Lernaufgabe in informatiktypischer Manier sehr präzise definiert werden kann.

> Eine *Lernaufgabe* wird definiert durch eine Beschreibung der dem lernenden System zur Verfügung stehenden Eingaben (ihrer Art, Verteilung, Eingabezeitpunkte, Darstellung und sonstigen Eigenschaften), der vom lernenden System erwarteten Ausgaben (ihrer Art, Funktion, Ausgabezeitpunkte, Darstellung und sonstigen Eigenschaften) und den Randbedingungen des Lernsystems selbst (z.B. maximale Laufzeiten oder Speicherverbrauch).

Innerhalb einer jeden so definierten Lernaufgabe ist also genau spezifiziert, was es bedeutet, wenn ein System lernt: es lernt erfolgreich genau dann, wenn es in der Lage ist, bei Eingaben, die den Spezifikationen entsprechen, unter den geforderten Randbedingungen Ausgaben mit den gewünschten Eigenschaften zu erzeugen.

Verwenden wir die bekannteste und am häufigsten betrachtete Lernaufgabe, das *Lernen aus klassifizierten Beispielen* zur ersten (noch unpräzisen und idealisierten) Illustration. Bei dieser Lernaufgabe wird angenommen, dass es eine unbekannte Funktion (nennen wir sie f) gibt, die Objekten einer bestimmten Grundmenge einen Zielwert zuordnet. Die Objektmenge könnte beispielsweise alle Kreditkartentransaktionen eines bestimmten Anbieters sein, und die gesuchte Funktion ordnet jeder Transaktion entweder den Wert "in_ordnung" oder den Wert "betrügerisch" zu. Dem lernenden System wird nun eine Menge von *Beispielen* der gesuchten Funktion zur Verfügung gestellt. Jedes Beispiel besteht erstens aus einer Objektbeschreibung, und zweitens dem Zielwert, den f diesem Objekt zuordnen würde. Lernen bedeutet in dieser Klasse von Lernaufgaben, das das lernende System mit den gegebenen Beispielen als Eingabe eine Funktionsbeschreibung h (die "Hypothese") ausgibt, mit der bei zukünftig vorgelegten Instanzen, also Objektbeschreibungen mit noch unbekanntem Zielwert, der Zielwert von f möglichst gut vorhergesagt werden kann.

Wir werden auf diese und andere Lernaufgaben im Verlauf des Kapitels noch genauer eingehen, ihre verschiedenen Präzisierungen und entsprechende Lernverfahren kennenlernen, und dabei also das Gebiet des maschinellen Lernens sozusagen *en passant* extensional definieren. Dabei bleibt die Schwäche jeder extensionalen Definition natürlich erhalten: dieses Kapitel kann nicht alle bekannten Lernaufgaben behandeln, und die Weiterentwicklung der Forschung im maschinellen Lernen ist gerade auch durch die Neu- und Redefinition

von Lernaufgaben gekennzeichnet.

1.1.3 Motivationen und Anwendungen

Je nach Motivation der Forschenden für die computergestützte Beschäftigung mit Lernphänomenen ergeben sich unterschiedliche Blickrichtungen auf das Gebiet. Aus der kognitionsorientierten Perspektive ist man daran interessiert, mit Hilfe von Computermodellen das menschliche Lernen besser zu verstehen und abzubilden. Aus der theoretischen Perspektive ist man daran interessiert, grundsätzlich zu verstehen, welche Lernaufgaben für welche lernenden Systeme mit welchem Aufwand prinzipiell zu bewältigen sind. Aus der algorithmisch-anwendungsorientierten Perspektive schließlich ist man daran interessiert, Verfahren zu entwickeln und Systeme zu konstruieren, die in praktisch relevanten Anwendungen Lernaufgaben lösen und einen Nutzen erbringen. Dabei sind diese Perspektiven keineswegs völlig disjunkt. Es gibt im Gegenteil viele Arbeiten im maschinellen Lernen, bei denen praktisch erfolgreiche Verfahren zunächst aus kognitiver Perspektive entwickelt worden sind und schließlich auch zu entsprechenden formalen Modellen führten.

Bei der praktischen Anwendung des maschinellen Lernens lassen sich (mindestens) zwei große Bereiche unterscheiden. Da ist zum einen die interaktive Analyse von vorher gesammelten Datenbeständen mit Hilfe von Lernverfahren. Dieser Anwendungstyp hat in den letzten Jahren eine sehr hohe Aufmerksamkeit erfahren, nicht zuletzt im Data Mining bzw. bei der Wissensentdeckung in Datenbanken (engl. knowledge discovery in databases), auf die wir im folgenden Abschnitt etwas genauer eingehen. Genau so wichtig ist aber die Verwendung von Lernverfahren zur Erzielung adaptiven Verhaltens, d.h., ihrer Online-Nutzung in einem Performanzsystem. Die verschiedenen Abschnitte dieses Kapitels enthalten jeweils Anwendungsbeispiele aus diesen Bereichen für die verschiedenen Lernaufgaben.

1.1.4 Wissensentdeckung

Über die Definitionen der Begriffe Data Mining bzw. Knowledge Discovery in Databases (KDD) herrscht derzeit weitgehend Einigkeit. Im wissenschaftlichen Bereich wird die folgende Definition des Begriffs KDD oft zitiert [Fayyad *et al.*, 1996]:

> Wissensentdeckung in Datenbanken ist der nichttriviale Prozeß der Identifikation gültiger, neuer, potentiell nützlicher und schlußendlich verständlicher Muster in (großen) Datenbeständen.

Data Mining wird dabei als Bezeichnung für den eigentlichen Analyseschritt, in dem Hypothesen gesucht und bewertet werden, verwendet, d.h. Data Mining ist ein Teilschritt des KDD-Prozesses. Im kommerziellen Bereich wird diese Unterscheidung oft nicht getroffen, hier wird der Begriff Data Mining in der Regel als Synonym von bzw. anstatt des (vielleicht zu unhandlich langen) Begriffs "Knowledge Discovery in Databases" verwendet. Im Deutschen wird in der Regel die Bezeichung "Wissensentdeckung" (WED) oder einfach ebenfalls "Data Mining" verwendet [Wrobel, 1998].

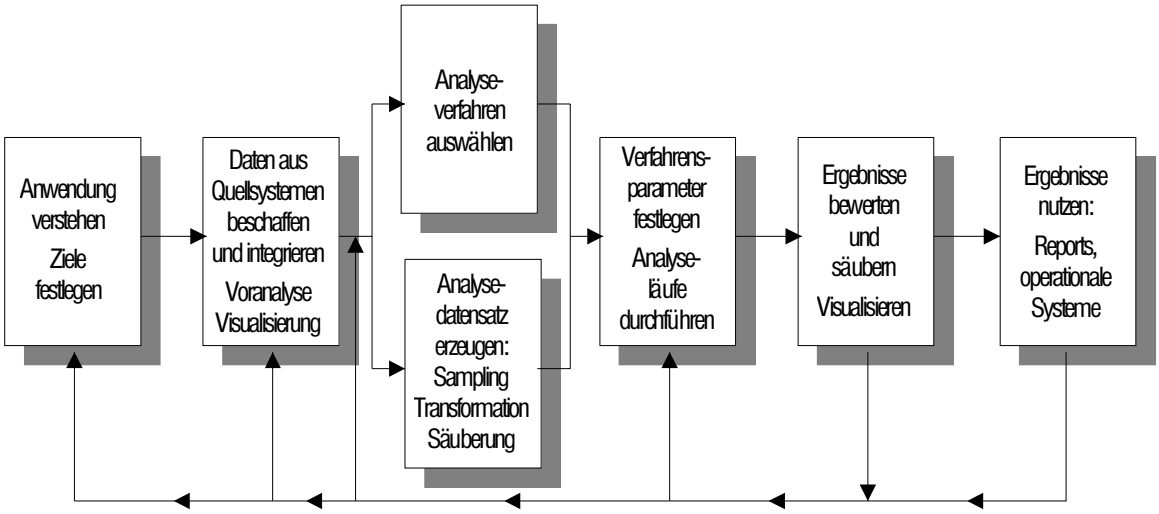

Abbildung 1.1: Der KDD-Prozess

Die Betonung des Prozeßaspektes in der Definition verweist auf die umfassende Sichtweise, die KDD auf den Prozeß der Datenanalyse hat: es werden alle Schritte von der ersten Beschäftigung mit einer Domäne bis hin zur Verwendung der Ergebnisse in Reports oder installierten Softwaresystemen betrachtet (Abbildung 1.1).

Was unterscheidet nun KDD vom Maschinellen Lernen? Zunächst einmal beinhaltet das Maschinelle Lernen grosse Bereiche, die bei KDD nicht betrachtet werden, insbesondere die Nutzung von Lernverfahren zur Erzielung von Adaptivität. Diese "Hälfte" des Maschinellen Lernens ist bei KDD normalerweise nicht von Interesse (ebensowenig wie Modelle menschlicher Lernprozesse). Gemeinsam ist beiden Gebieten das Interesse an der Analyse bereits gesammelter Daten, und tatsächlich kommen im Analyseschritt zu grossen Teilen Techniken des Maschinellen Lernens zum Einsatz. Der Bereich der Wissensentdeckung hat andererseits das Maschinelle Lernen um einige interessante deskriptive Lernaufgaben bereichert, wie z.B. die Subgruppenentdeckung (Abschnitt 1.11) oder das Lernen von Assoziationsregeln (Abschnitt 1.10).

Ein weiterer Unterschied zwischen KDD und ML liegt in den Anforderungen an die Skalierbarkeit der Verfahren. Hier betont KDD sehr viel stärker als das "klassische" ML die Notwendigkeit, mit "großen" Datenmengen umzugehen. Dabei bezieht sich Größe sowohl auf die Anzahl der Variablen (Spalten der einzelnen Tabellen), wo einige Hundert bis einige Tausend angestrebt werden, als auch auf die Anzahl der Datenpunkte (Zeilen), wo die Ziele im Millionen- bis Milliardenbereich liegen. So ist denn auch der Datenbankbereich neben der Statistik eine der wichtigsten zu KDD beitragenden Nachbardisziplinen.

Auch wenn in der KDD der interaktive und iterative Prozess, bei dem Mensch und Lernverfahren gemeinsam verständliches und interessantes Wissen entdecken, sehr stark betont wird, so heißt das nicht, das diese Sicht im Maschinellen Lernen nicht ebenso verbreitet war; KDD hat hier jedoch zu einer deutlichen Explizierung beigetragen.

ID	Feuchte	Säure	Temp.	Klasse
1	trocken	basisch	7	W
2	feucht	neutral	8	N
3	trocken	neutral	7	W
4	feucht	alkalisch	5	N
5	trocken	neutral	8	N
6	trocken	neutral	6	W
7	trocken	neutral	11	N
8	trocken	neutral	9	N

ID	Feuchte	Säure	Temp.	Klasse
9	trocken	alkalisch	9	W
10	trocken	alkalisch	8	W
11	feucht	basisch	7	N
12	feucht	neutral	10	W
13	trocken	basisch	6	W
14	feucht	alkalisch	7	N
15	trocken	basisch	3	N
16	trocken	basisch	4	W

Tabelle 1.1: Beispiele für Standorte einer Pflanzenart

1.2 Funktionslernen aus Beispielen

Das Funktionslernen aus Beispielen ist unbestritten die populärste, am häufigsten betrachtetste und auch am häufigsten genutzte Lernaufgabe des Maschinellen Lernens. Betrachten wir eine Beispielanwendung aus dem Bereich der Pflanzenökologie [Kirsten *et al.*, 1998]. Für das wissenschaftliche Verständnis der Pflanzenwelt, aber auch für praktische Zwecke, z.B. für die Auswahl von Pflanzen für die Wiederaufforstung von Flächen, ist es von großer Bedeutung, genau zu verstehen, unter welchen boden- und lokalklimatischen Bedingungen eine bestimmte Pflanzenart wachsen kann. Gesucht wird also ein Klassifikator, der für einen bestimmten Standort entscheiden kann, ob z.B. die Rotbuche an diesem Standort wachsen kann. Um solche Klassifikatoren manuell aufzustellen, bereisen Ökologen unterschiedliche Standorte, halten die Eigenschaften des Bodens und des Lokalklimas fest und stellen fest, ob eine bestimmte Pflanze an diesem Standort wachsen konnte oder nicht. So entstehen also für jede Pflanzenart *Beispiele* von Standorten, an denen diese Pflanze wachsen kann, und Beispiele von Standorten, an denen diese Pflanze nicht wachsen kann[1].

Tabelle 1.1 zeigt einige Beispiele solcher Standorte für die Rotbuche[2]. Jede Zeile dieser Tabelle stellt dabei einen Standort dar. In der ersten Spalte finden wir einen numerischen Bezeichner für den Standort. Das Merkmal Bodenfeuchte kann die Werte trocken oder feucht annehmen, das Merkmal Bodensäure die Werte basisch, neutral oder sauer, und das Merkmal Durchschnittstemperatur die gemessene Durchschnittstemperatur auf eine Stelle gerundet. Die letzte Spalte der Tabelle gibt die Klassifikation des Standortes an. Mit W sind diejenigen Standorte gekennzeichnet, an denen die Rotbuche wachsen kann, mit N sind diejenigen Standorte gekennzeichnet, an denen sie nicht wachsen kann.

Sollen nun die Ökologen bei der aufwendigen Formulierung der Klassifikatoren unterstützt werden, so handelt es sich dabei um die Lernaufgabe "Funktionslernen aus Beispielen". Wir gehen nämlich implizit davon aus, daß es eine uns und dem Lernverfahren

[1] Es wird geschätzt, daß allein in Deutschland in diesem Jahrhundert einige hunderttausend Standortansprachen gesammelt worden sind. Tatsächlich kann durch eine Standortbegehung nur festgestellt werden, daß eine bestimmte Pflanze an einem bestimmten Standort wächst, da natürlich nicht alle Pflanzen, die potenziell dort wachsen könnten, tatsächlich auch anzutreffen sind. Zur Vereinfachung unseres Beispiels betrachten wir diese Problematik hier nicht.

[2] Das Beispiel ist fiktiv, stark vereinfacht und dient nur zur Illustration. Es sollte nicht als akkurate Beschreibung ökologischer Zusammenhänge gelesen werden.

nicht bekannte Klassifikationsfunktion f gibt, die für jeden Standort die "wahre" Klassifikation ausgibt. Die Beispiele in unserer Tabelle stellen also eine *Stichprobe* des Verhaltens dieser Funktion f dar, die in unserem Fall durch die Gesetze der Biologie bestimmt wird. Den gesuchten Klassifikator können wir als eine Funktion h ansehen, welche idealerweise für jeden möglichen Standort genau das gleiche Ergebnis liefern sollte wie die uns unbekannte Funktion f. Ein möglicher Klassifikator h (in Regelform) könnte beispielsweise sein:

> "Wenn Bodenfeuchte=trocken und Bodensäure=neutral, dann Klasse=W, sonst Klasse=N."

Dieser Klassifikator liefert allerdings bereits für die Beispiele nicht genau die gleichen Ergebnisse wie die unbekannte Funktion f, so daß wir ihn wohl nicht als wünschenswertes Lernergebnis ansehen würden.

Wir präzisieren daher die Lernaufgabe "Funktionslernen aus Beispielen" wie folgt.

Definition 1.2.1 (Funktionslernen aus Beispielen) *Es sei X eine Menge möglicher* Instanzenbeschreibungen*, D eine Wahrscheinlichkeitsverteilung auf X, und Y eine Menge möglicher* Zielwerte*. Es sei weiterhin H eine Menge zulässiger Funktionen (auch als* Hypothesensprache *L_H bezeichnet). Eine Lernaufgabe vom Typ* Funktionslernen aus Beispielen *sieht dann wie folgt aus.*

Gegeben:

- *Eine Menge E von* Beispielen *der Form $(x, y) \in X \times Y$, für die gilt: $y = f(x)$ für eine unbekannte Funktion f.*

Finde: *Eine Funktion $h \in H$*

- *so daß der Fehler $error_D(h, f)$ von h im Vergleich zu f bei gemäß der Verteilung D gezogenen Instanzen aus X möglichst gering ist.*

Beispiele der gleichen Form (x, y) können auch mehrfach in E vorkommen. Trotzdem spricht man von E meist als Menge, da jedes Beispiel einem unterschiedlichen Ereignis oder Objekt in der Welt entspricht. Auch die Rolle der Verteilung D bedarf weiterer Erläuterungen. So wäre es doch viel einfacher gewesen zu fordern, daß der gelernte Klassifikator h möglichst viele der gegebenen Beispiele genauso klassifiziert wie die unbekannte Funktion f. Dieses Gütemaß für eine gelernte Funktion h bezeichnet man als *Trainingsfehler*, da die gegebenen Beispiele E oft auch als *Trainingsmenge* bezeichnet werden.

Definition 1.2.2 (Trainingsfehler) *Seien E, f und H wie in der vorhergehenden Definition. Als* Trainingsfehler *von h bezüglich f bezeichnet man*

$$error_E(h) := \sum_{(x,y)\in E} error(h(x), y)$$

wobei man üblicherweise für numerische Zielwerte Y definiert:

$$error(h(x), y) := (h(x) - y)^2$$

und für diskrete Zielmengen Y mit nur wenigen Werten:

$$error(h(x), y) := 0, \text{ wenn } h(x) = y, \text{ sonst } 1$$

Ersteren bezeichnet man als den quadrierten Fehler*, letzteren als* 0-1-loss*.*

Der Trainingsfehler misst also, wie gut unsere gelernte Funktion h die bereits bekannten Objekte klassifiziert. Er ist daher einfach auszuwerten, doch tatsächlich ist man naheliegenderweise mehr daran interessiert, wie gut die gelernte Funktion bei zukünftigen, noch nicht bekannten Objekten mit der wahren, aber unbekannten Funktion f übereinstimmt. Um dies sinnvoll definieren zu können, müssen wir wenigstens annehmen, daß die uns zukünftig präsentierten, zu klassifizierenden Objekte aus den einzelnen Bereichen aus dem Instanzenraum X mit ähnlicher Häufigkeit gewählt werden wie dies bei den Beispielen E der Fall war. Gilt dies nicht, so kann man sich leicht vorstellen, daß unsere Lernaufgabe sehr schwer bzw. unlösbar wird. Wir nehmen genau deshalb an, daß es die oben eingeführte Wahrscheinlichkeitsverteilung D gibt, die die Auswahlwahrscheinlichkeit bestimmter Instanzen aus X sowohl für die Beispiele als auch für zukünftig zu klassifizierende Objekte angibt. Mit Hilfe von D können wir nun auch den eigentlich zu minimierenden wahren Fehler definieren.

Definition 1.2.3 (Wahrer Fehler) *Seien D, f und H wie in den vorangegangenen Definitionen. Als* wahren Fehler *der Hypothese h bezeichnet man:*

$$error_D(h, f) := E_D[error(h(x), f(x)]$$

also den erwarteten (durchschnittlichen) Fehler, falls Instanzen aus X gemäß D gewählt werden.

Da ein Lernverfahren D und f selbst nicht kennt, sondern nur die mit D gewählten Beispiele E, ist der wahre Fehler nicht direkt zu bestimmen. Dennoch müssen Lernverfahren sicherstellen, daß sie nicht einfach den Trainingsfehler minimieren, denn dabei tritt das Phänomen der sogenannten *Überanpassung* (engl. *overfitting*) auf. Überanpassung bedeutet, daß ein Lernverfahren eine Hypothese auswählt, die zwar auf den Trainingsdaten einen minimalen Fehler hat, jedoch auf späteren zu klassifizierenden Daten einen sehr viel höheren wahren Fehler aufweist als andere ebenfalls mögliche Hypothesen. Das Lernverfahren hat sich also den verfügbaren Trainingsbeispielen sehr genau angepaßt, dabei jedoch die zugrunde liegende wahre Funktion f nicht optimal erfaßt.

Dieses Phänomen tritt wohlgemerkt nicht nur dann auf, wenn die Beispieldaten *verrauscht* sind, also durch fehlerhafte Eingaben Beispiele enthalten, bei denen $y \neq f(x)$. Auch wenn die Beispielmenge fehlerfrei der gesuchten Funktion f entspricht, kann es bei genügend großem oder ungünstig gestaltetem Hypothesenraum L_H Hypothesen geben, die zufällig die gesehenen Beispiele richtig klassifizieren, von der gesuchten Funktion f aber ansonsten weit entfernt sind. Alle im folgenden vorgestellten Lernverfahren enthalten daher Mechanismen, um das Überanpassungsphänomen vermeiden zu helfen (siehe z. B. die Nutzung einer Validierungsmenge, Abschnitt 1.3.1, oder die Kreuzvalidierung, Abschnitt 1.4.2). Allgemein kann man solche Ansätze als *Modellselektionsverfahren* sehen, und die aktuelle Forschung verwendet auch eher den allgemeinen Begriff der Modellselektion als den Begriff der Überanpassung (siehe z.B. [Kearns *et al.*, 1997; Scheffer und Joachims, 1999], auch mit Hinweisen auf andere Modellselektionsansätze). Weitere sehr populäre allgemeine Ansätze zur Steigerung der Genauigkeit eines Lernverfahrens beruhen auf Kombinationen verschiedener gelernter Modelle (*bagging* [Breiman, 1996], *boosting* [Schapire, 1999]).

Definition 1.2.1 beschreibt nur einen Spezialfall des Funktionslernens aus Beispielen. Es wird die Annahme gemacht, dass es eine Funktion f gibt, welche für die gleiche Instanzenbeschreibung x immer den gleichen Zielwert $y = f(x)$ liefert. Der Zusammenhang ist also deterministisch. Realistischer ist es meist, den Zusammenhang durch eine bedingte Wahrscheinlichkeitsverteilung $P(Y|X)$ zu modellieren. Dann geht nicht nur $D = P(X)$, sondern auch $P(Y|X)$ in den wahren Fehler ein. Analog zu f und $P(X)$ in Definition 1.2.1, beschreibt im allgemeinen Fall $P(X, Y) = P(Y|X)P(X)$ das Lernziel vollständig.

Abschließend sei noch erwähnt, daß sich für bestimmte Ausprägungen der Lernaufgabe Funktionslernen aus Beispielen besondere Namen eingebürgert haben. Besteht die Zielmenge Y aus einer Menge mit zwei diskreten Werten, spricht man auch von *Begriffslernen aus Beispielen*, bei dem alle Objekte mit einer bestimmten Ausprägung von Y als "Instanzen" (Mitglieder) und alle Objekte mit der anderen Ausprägung als "Nicht-Instanzen" (Nicht-Mitglieder) bezeichnet und die gesuchte Funktion als "Begriff" bezeichnet werden. Bei unserer Beispielanwendung zum Pflanzenwuchs handelt es sich also um Begriffslernen aus Beispielen. Hat Y mehr als zwei, aber nur sehr wenige Werte, so spricht man auch vom Lernen von Klassifikationen. Ist Y dagegen numerisch, so bezeichnet man die Funktionslernaufgabe oft auch als *Regression*. Wie wir in Abschnitt 1.7 zur Induktiven Logikprogrammierung sehen werden, lassen sich auch Klassifikations- und Regressionsaufgaben auf binäre Begriffslernprobleme abbilden, indem man nicht eine Funktion lernt, die eine Instanz x auf einen Zielwert y abbildet, sondern eine Funktion, die das Paar (x, y) genau dann auf den Wert 1 (bzw. true) abbildet, wenn $y = f(x)$, und sonst auf 0 (bzw. false).

1.2.1 Lernen als Suche

Bei der Lernaufgabe "Funktionslernen aus Beispielen" - und auch bei anderen Lernaufgaben - ist es oft nützlich, sich den Lernprozeß als Suche nach einer geeigneten Hypothese h innerhalb des zur Verfügung stehenden Hypothesenraums L_H vorzustellen [Mitchell, 1982]. Dadurch kann man verschiedene Lernverfahren leicht anhand der Suchstrategie, die sie innerhalb dieses Suchraumes verfolgen, einordnen. Insbesondere lassen sich folgende grobe Klassen von Suchstrategien unterscheiden.

Geordnete Suche In bestimmten Hypothesenräumen L_H ist es möglich, die Hypothesen entlang einer sogenannten *Verallgemeinerungsrelation* anzuordnen (siehe auch Abschnitt 1.7). Betrachten wir dazu noch einmal unsere ökologische Beispielanwendung. Es sei nun unsere Hypothese h_1 die Regel "Wenn Bodensäure = neutral, dann Klasse = W.". Betrachten wir als zweite Hypothese h_2 die Regel "Wenn Bodensäure = neutral und Bodenfeuchte = trocken, dann Klasse = W.". Man kann sich nun überlegen, daß alle Standorte, die mit h_2 der Klasse W zugeordnet werden können, auch mit h_1 der Klasse W zugeordnet werden können. h_1 ist *allgemeiner* als h_2. Kann man für einen Hypothesenraum eine solche Allgemeinheitsrelation definieren, so können Lernverfahren ihre Suche effizienter gestalten, indem sie beispielsweise mit der allgemeinsten Hypothese beginnen, und diese dann sukzessive durch weniger allgemeine (*speziellere*) Hypothesen ersetzen. Dieses ist besonders schön bei den in Abschnitt 1.7

beschriebenen Verfahren der Induktiven Logikprogrammierung zu sehen.

Gradientensuche Bei vielen Lernverfahren bestehen die Hypothesenräume aus numerischen Funktionen, die bei vorgegebener Form durch eine feste Anzahl numerischer Parameter bestimmt werden. Das einfachste Beispiel hierfür sind *lineare Funktionen*, die auf der Basis von numerischen Merkmalen $x_1, \ldots, x_n$ einen numerischen Wert y wie folgt vorhersagen:

$$y = h(x) = b_0 + b_1 x_1 + \cdots + b_n x_n$$

Für viele Klassen solcher Funktionen läßt sich der Gradient (die partielle Ableitung, die "Steigung") des Fehlers bezüglich der Funktionsparameter berechnen. Ausgehend von einer anfänglichen Hypothese kann das Lernverfahren dann die Parameter der Hypothese jeweils in Richtung der stärksten Fehlerverringerung verändern. Eine solche Suche führt beispielsweise der *Backpropagation*-Algorithmus für neuronale Netze aus [Rumelhart *et al.*, 1986], der in Kapitel beschrieben wird. Auch die in Abschnitt 1.5 beschriebene Stützvektormethode nutzt die numerischen Eigenschaften des Hypothesenraumes in einem iterativen Verfahren aus[3].

REFERENZ AUF ANDERES BUCHKAPITEL

Stochastische Suche Bei der stochastischen Suche werden im Prinzip keine Annahmen über Struktur und Eigenschaften des Hypothesenraumes L_H gemacht. Stattdessen geht man davon aus, daß dem Lernverfahren eine Reihe von *Operatoren* zur Verfügung stehen, die in der Lage sind, aus einer oder mehreren bestehenden Hypothesen neue Hypothesen zu erzeugen. Dabei wird normalerweise darauf geachtet, daß das Verfahren sowohl Operatoren zur Verfügung hat, die kleine Veränderungen erzeugen, als auch solche, die große Sprünge im Hypothesenraum machen können. Die Suche im Hypothesenraum geschieht dann, ausgehend von einer oder mehreren Anfangshypothesen, durch geeignete stochastische Auswahl der jeweils anzuwendenden Operatoren. So wird beim sogenannten *simulierten Ausglühen* ([Kirkpatrick *et al.*, 1983], engl. *simulated annealing*) die anfangs hohe Wahrscheinlichkeit für die Auswahl weit springender Operatoren im Laufe des Lernlaufs immer weiter reduziert, um so schließlich zu einer bestimmten Hypothese konvergieren zu können. Ein weiteres populäres Beispiel der operatorgestützten Suche sind die *genetischen Algorithmen* [Goldberg und Holland, 1988], bei denen eine Population von n Hypothesen durch Operatoren evolviert wird, die durch die Biologie inspiriert wurden (Selektion, Mutation, Crossover).

Nach der Lektüre der Beschreibung der unterschiedlichen Lernverfahren in den folgenden Abschnitten ist es eine nützliche Übung, sich klarzumachen, welche Eigenschaften des Hypothesenraums von dem jeweiligen Lernverfahren ausgenutzt werden, und mit welcher Suchstrategie es seine Ausgabehypothese produziert.

1.3 Entscheidungsbäume

Entscheidungsbaumverfahren [Breiman *et al.*, 1984; Quinlan, 1993; Murthy, 1998] sind wahrscheinlich zur Zeit die populärsten und am häufigsten benutzten Verfahren des ma-

[3] Für lineare Funktionen läßt sich das Fehlerminimum im übrigen algebraisch bestimmen, so daß man dort keine Gradientensuche benötigt.

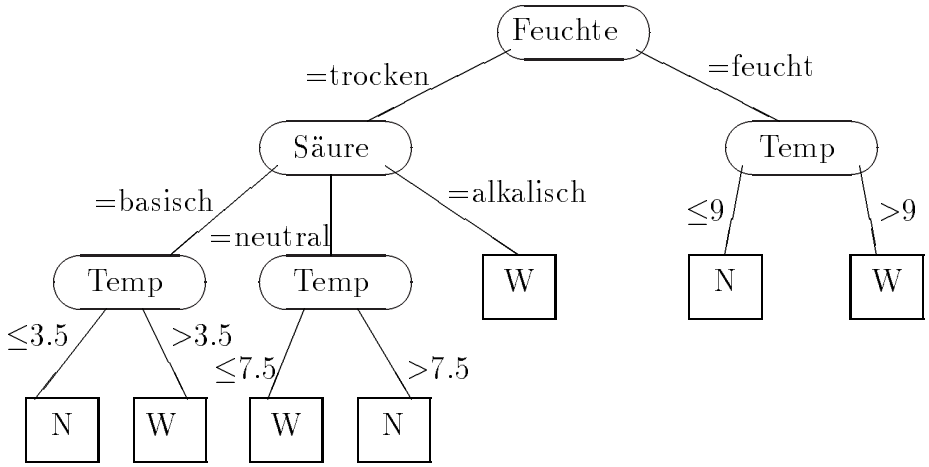

Abbildung 1.2: Entscheidungsbaum zur Standortbeurteilung

schinellen Lernens. Dies liegt vermutlich daran, daß sie einfach zu bedienen sind, nur relativ kurze Laufzeiten benötigen und daß die produzierten Entscheidungsbäume für die Benutzer relativ einfach zu verstehen sind. Betrachten wir als Beispiel einen Entscheidungsbaum, mit dem man die im vergangenen Abschnitt vorgestellte Aufgabe der Klassifikation von Standorten hinsichtlich ihrer Eignung für die Rotbuche vornehmen könnte.

Der in Abbildung 1.2 gezeigte Entscheidungsbaum ist von oben nach unten zu lesen. Zur Klassifikation eines neuen Standortes beginnt man an der Wurzel des Baumes und prüft zunächst den Wert des in diesem Knoten angegebenen Attributs. Man verfolgt dann den Pfad zum nächsten Knoten, der dem in der zu klassifizierenden Instanz vorgefundenen Wert des Attributs entspricht. Dadurch gelangt man zu einem neuen Knoten, wo das nächste zu prüfende Attribut wartet und so weiter. Schließlich gelangt man zu einem Blattknoten, in dem keine weitere Frage, sondern die auszugebende Klasse steht. Bei dem in der Abbildung gezeigten Entscheidungsbaum würde also der Standort $< trocken, alkalisch, 7 >$ vom Wurzelknoten aus zunächst in den linken Ast wandern, um dann nach rechts zu verzweigen und somit als "W" klassifiziert zu werden.

Dieses Beispiel zeigt bereits die Lernaufgabe, die von den meisten aktuellen Entscheidungsbaumverfahren gelöst wird.

Definition 1.3.1 (Entscheidungsbaumlernen) *Entscheidungsbaumverfahren lösen die Lernaufgabe Funktionslernen aus Beispielen, wobei X eine Menge von durch n numerische oder diskrete Attribute beschriebenen Instanzen ist, Y eine kleine Menge von diskreten Klassenwerten und L_H die Menge der aus diesen Attributen und ihren Werten konstruierbaren Entscheidungsbäumen ist*[4].

Wie in unserem Beispiel werden als Bedingungen an den Knoten und Pfaden des Baumes in der Regel die Operatoren $=$, $\leq$ und oft auch $\in$ (aktueller Wert muß Element einer angegebenen Wertemenge sein) zugelassen.

[4]Es gibt auch Entscheidungsbaumverfahren, die komplexere Instanzenräume X beziehungsweise auch numerische Wertemengen Y verarbeiten können. Wir konzentrieren uns hier auf die am häufigsten anzutreffende Variante der Lernaufgabe.

naiv		
ID	K	V
1	W	N
2	N	N
3	W	N
4	N	N
5	N	N
6	W	N
7	N	N
8	N	N
9	W	N
10	W	N
11	N	N
12	W	N
13	W	N
14	N	N
15	N	N
16	W	N

nach Feuchte		
ID	K	V
trocken		
1	W	W
3	W	W
5	N	W
6	W	W
7	N	W
8	N	W
9	W	W
10	W	W
13	W	W
15	N	W
16	W	W
feucht		
2	N	N
4	N	N
11	N	N
12	W	N
14	N	N

nach Säure		
ID	K	V
basisch		
1	W	W
11	N	W
13	W	W
15	N	W
16	W	W
neutral		
2	N	N
3	W	N
5	N	N
6	W	N
7	N	N
8	N	N
12	W	N
alkalisch		
4	N	N
9	W	N
10	W	N
14	N	N

nach Temperatur		
ID	K	V
≤ 6.5		
15	N	W
16	W	W
4	N	W
6	W	W
13	W	W
> 6.5		
11	N	N
3	W	N
14	N	N
1	W	N
5	N	N
2	N	N
10	W	N
8	N	N
9	W	N
12	W	N
7	N	N

Tabelle 1.2: Beispiele nach verschiedenen Kriterien sortiert. Die ID entsprechen denen aus Tabelle 1.1. Die Spalten "K" zeigen die Klasse des Beispiels und die Spalten "V" die Vorhersage.

Da in jedem Pfad eines Entscheidungsbaumes die Attribute in unterschiedlicher Reihenfolge vorkommen können (numerische Attribute sogar mehrmals in einem Pfad), ist die Anzahl der möglichen Entscheidungsbäume exponentiell in der Anzahl der Attribute. Es ist daher in der Regel nicht praktikabel, alle möglichen Entscheidungsbäume zu betrachten und zu vergleichen. Stattdessen wird der Baum von der Wurzel her (top-down, daher auch *top-down induction of decision trees*, TDIDT, genannt) aufgebaut, indem man sich — zunächst für den Wurzelknoten — überlegt, welches Attribut lokal in dem jeweiligen Knoten die größte Verbesserung der Klassifikationsleistung erbringen würde. Wir können das Prinzip am einfachsten anhand unserer Beispielanwendung illustrieren.

Tabelle 1.1 aus dem vorangegangenen Abschnitt zeigt die Beispiele, aus denen wir nun den Entscheidungsbaum bauen wollen. Da wir noch gar keine Verzweigung vorgenommen haben, ist es am günstigsten, für alle Beispiele die Mehrheitsklasse vorhersagen, in unserem Fall (beide Klassen sind gleich häufig) also z.B. "N". Damit erreichen wir einen Trainingsfehler von $\frac{8}{16}$. Diese naive Vorhersage zeigt der linke Kasten in Tabelle 1.2. Teilen wir nun probeweise unsere Standorte anhand des Attributes Bodenfeuchte gemäß der auftretenden Werte in zwei Gruppen auf, und sagen dann für diese beiden Gruppen jeweils deren Mehrheitsklasse voraus, so ergibt sich das in dem zweiten Kasten von Tabelle 1.2 dargestellte Bild.

In der Gruppe der Standorte, bei denen Bodenfeuchte gleich trocken war (11 Standorte)

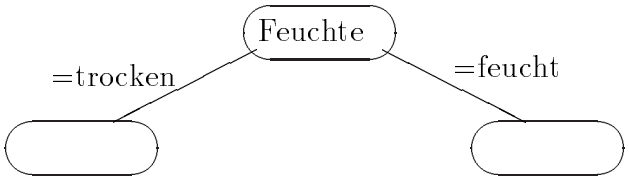

Abbildung 1.3: Entscheidungsbaum nach Bestimmung des Wurzelattributs

erreichen wir so eine Fehlerrate von $\frac{4}{11}$. In der anderen Gruppe (5 Standorte) erreichen wir eine Fehlerrate von $\frac{1}{5}$. Insgesamt ergibt dies also jetzt eine Trainingsfehlerrate von $\frac{11}{16}\frac{4}{11} + \frac{5}{16}\frac{1}{5} = \frac{5}{16}$. Wir können also feststellen, daß wir mit der Aufteilung der Beispielmenge anhand des Attributs Bodenfeuchte unseren Trainingsfehler verbessern können. Möglicherweise ist jedoch die Verbesserung durch eines der anderen Attribute noch stärker. Wir prüfen dies zunächst für das Attribut Bodensäure und erhalten die Situation in dem dritten Kasten von Tabelle 1.2.

Wie man sieht, können wir nach Aufteilung mit diesem Attribut in den jeweiligen Gruppen 2 von 5, 3 von 7 und 2 von 4 Standorten richtig klassifizieren, erreichen also insgesamt einen Trainingsfehler von $\frac{5}{16}\frac{2}{5} + \frac{7}{16}\frac{3}{7} + \frac{4}{16}\frac{2}{4} = \frac{7}{16}$. Mit diesem Attribut ist also eine weniger große Verbesserung des Trainingsfehlers möglich. Das dritte Attribut Durchschnittstemperatur ist numerisch, so daß es nicht sinnvoll ist, anhand der einzelnen Werte aufzuteilen. Wir teilen die Standorte stattdessen anhand eines Schwellwertes in zwei Gruppen auf. Da wir einen Schwellwert jeweils in die Mitte zwischen zwei tatsächlich aufgetretene Werten legen können, ergeben sich daraus in unserem Beispiel (9 verschiedene Werte) 8 mögliche Aufteilungen in 2 Gruppen.

Der rechte Kasten in Tabelle 1.2 zeigt die Aufteilung für den Schwellwert $6,5$, welche von allen 8 möglichen Schwellwerten den günstigsten Trainingsfehler erreicht, nämlich $\frac{5}{16}\frac{2}{5} + \frac{11}{16}\frac{5}{11} = \frac{7}{16}$. Da auch dieser Wert schlechter ist als der für das Attribut Bodenfeuchte, entscheiden wir uns, unseren Entscheidungsbaum mit dem Attribut Bodenfeuchte zu beginnen. Der Entscheidungsbaum sieht also jetzt aus wie in Abbildung 1.3.

Durch das Einführen des ersten Knotens teilen sich, wie in Tabelle 1.2 schon gezeigt, unsere Beispiele nun auf die beiden Zweige des Baumes auf. Zum weiteren Aufbau des Baumes können wir daher nun die gleiche Prozedur rekursiv jeweils mit den nun kleineren Teilgruppen in den beiden neuentstandenen, noch leeren Knoten durchführen. Dies tun wir solange, bis wir in einem Knoten und seiner Teilgruppe einen Trainingsfehler von 0 erreicht haben (alle Instanzen sind von einer Klasse), bzw. bis uns die möglichen Vergleichsbedingungen für die Knoten ausgehen (denn schließlich ist es nicht sinnvoll, denselben Test mehrmals in einem Pfad zu wiederholen). In unserem Beispiel ergibt sich dabei die in Tabelle 1.3 dargestellte Gruppenaufteilung und der in Abbildung 1.2 dargestellte Entscheidungsbaum. Abbildung 1.4 enthält den allgemeinen Algorithmus zur Entscheidungbaumkonstruktion, in dem das Qualitätsmaß "Qualität" eine wichtige Rolle spielt. Wir haben in unserem Beispiel einfach die durchschnittliche Verringerung des Trainingsfehlers genommen. Tatsächlich hat sich herausgestellt, daß einige andere Bewertungsfunktionen für diesen Zweck geeigneter sind. Eine davon, der sogenannte *Informationsgewinn* ([Quinlan, 1993], engl. *information*

ID	Feuchte	Säure	Temp.	Klasse	Vorhersage
15	trocken	basisch	3	N	N
16	trocken	basisch	4	W	W
13	trocken	basisch	6	W	W
1	trocken	basisch	7	W	W
6	trocken	neutral	6	W	W
3	trocken	neutral	7	W	W
5	trocken	neutral	8	N	N
8	trocken	neutral	9	N	N
7	trocken	neutral	11	N	N
9	trocken	alkalisch	9	W	W
10	trocken	alkalisch	8	W	W
4	feucht	alkalisch	5	N	N
11	feucht	basisch	7	N	N
14	feucht	alkalisch	7	N	N
2	feucht	neutral	8	N	N
12	feucht	neutral	10	W	W

Tabelle 1.3: Endaufteilung mit Trainingsfehler 0

Sei E die Beispielmenge und *tests* die Menge aller aus den Attributen und Attributwerten der Beispielmenge konstruierbaren Knotentests.
TDIDT(E,Tests)

- Falls E nur Beispiele einer Klasse enthält, liefere einen Blattknoten mit dieser Klasse zurück. Andernfalls:
- Für jeden Test in Tests, berechne Qualität(Test,E)
- Wähle den Test mit der höchsten Qualität für den aktuellen Knoten aus.
- Teile E anhand dieses Tests in 2 oder mehr Teilmengen $E_1, \ldots, E_k$ auf.
- Für $i = 1, \ldots, k$ rufe rekursiv: $T_i := TDIDT(E_i, Tests \backslash \{test\})$
- Liefere als Resultat den aktuellen Knoten mit den darunter hängenden Teilbäumen $T_1, \ldots, T_k$ zurück.

Abbildung 1.4: Grundalgorithmus für Entscheidungsbaumverfahren

gain) soll hier nun kurz vorgestellt werden. In der Kodierungstheorie [Shannon und Weaver, 1969] bezeichnet man als Informationsgehalt einer bestimmten Menge von Symbolen die minimale Anzahl von Fragen, die man stellen muß, um ohne vorheriges Wissen ein Symbol dieser Menge eindeutig identifizieren zu können. Da die Symbole unterschiedlich wahrscheinlich sind, ist genauer gesagt die im Mittel zu erwartende Anzahl von Fragen gemeint. Falls die m Symbole einer Menge mit Wahrscheinlichkeiten $p_1, \ldots, p_m$ auftreten, so läßt sich der auch *Entropie* genannte Informationsgehalt, also die zu erwartende und in Bit gemessene Anzahl der notwendigen Fragen wie folgt berechnen:

$$I(p_1, \ldots, p_m) := \sum_{i=1}^{m} -p_i log p_i$$

Beim Bau eines Entscheidungsbaumes kennen wir die Wahrscheinlichkeiten der einzelnen Klassen nicht, können aber näherungsweise die relativen Häufigkeiten der Klassen im aktuellen Knoten verwenden. Statt desjenigen Tests, der den durchschnittlichen zu erwartenden Trainingsfehler am stärksten reduziert, verwenden wir beim Informationsgewinn-Kriterium denjenigen Test, der den durchschnittlichen Informationsgehalt der neu entstehenden Teilmengen am stärksten reduziert. Da der Informationsgehalt vor der Aufteilung für alle möglichen Tests gleich ist, brauchen wir diese Differenz nicht tatsächlich zu berechnen, sondern können einfach das Qualitätsmaß wie folgt definieren:

Definition 1.3.2 (Qualitätsmaß) *Sei $Test$ ein Test, der die Beispielmenge E in Teilmengen $E_1, \ldots, E_k$ zerlegt. Es seien $p_{i,1}, \ldots, p_{i,m}$ die relativen Häufigkeiten der m Klassen in der Teilmenge E_i. Dann definiere:*

$$Qualität(Test, E) := IG(Test, E) := -\sum_{i=1}^{k} \frac{|E_i|}{|E|} I(p_1, \ldots, p_m)$$

1.3.1 Stutzen des Baumes

Auch durch die Verwendung des Informationsgewinns als Qualitätsmaß ändert sich nichts daran, daß die Baumkonstruktion nur am Trainingsfehler orientiert ist, und erst endet, wenn der Trainingsfehler Null geworden ist. Hier besteht also ein großes Risiko der Überanpassung der Trainingsdaten. Bei Entscheidungsbaumverfahren wird deshalb üblicherweise der aufgebaute Baum in einem separaten Schritt *gestutzt*.

Abbildung 1.5 zeigt den Baum, der durch das Stutzen der Blätter ganz unten links entsteht. Da nun ein vorheriger innerer Knoten zum Blattknoten wird, finden sich in diesem Knoten Instanzen unterschiedlicher Klassen wieder, das heißt, der gestutzte Baum macht jetzt, wie sich Tabelle 1.3 entnehmen läßt, auf der Trainingsmenge einen Fehler (Beispiel 15 wird jetzt fälschlicherweise als "W" klassifiziert).

Warum ist diese Prozedur sinnvoll, und wieweit soll der Baum zurückgestutzt werden? Man kann zum einen argumentieren, daß Bäume geringerer Tiefe den Instanzenraum weniger fein unterteilen können, und daß bei diesen Bäumen daher das Risiko der Überanpassung geringer ist. Kleinere Bäume sind auch aus Gründen der Verständlichkeit vorzuziehen. Andererseits ist klar, daß spätestens dann, wenn nur noch der Wurzelknoten übrig bleibt, der Baum vermutlich nicht nur einen hohen Trainings-, sondern auch einen hohen wahren Fehler haben wird.

Man braucht also eine Methode, um zu schätzen, wie sich der wahre Fehler des Baumes beim Zurückstutzen entwickelt. Dann kann man den Baum soweit zurückstutzen, wie der wahre Fehler sinkt. Eine allgemeine Methode, den wahren Fehler zu schätzen, besteht darin, einen bestimmten Anteil der verfügbaren Trainingsbeispiele nicht zum Training zu verwenden, sondern ihn als *Validierungsmenge* zunächst beiseite zu legen. Da der Entscheidungsbaum unabhängig von dieser Validierungsmenge erzeugt wird, kann der gemessene Fehler auf der Validierungsmenge als Schätzer für den wahren Fehler verwendet werden.

Sind nur wenige Trainingsbeispiele vorhanden, würde das Beiseitelegen einer Validierungsmenge möglicherweise nicht mehr genügend Daten übrig lassen, um sinnvoll den un-

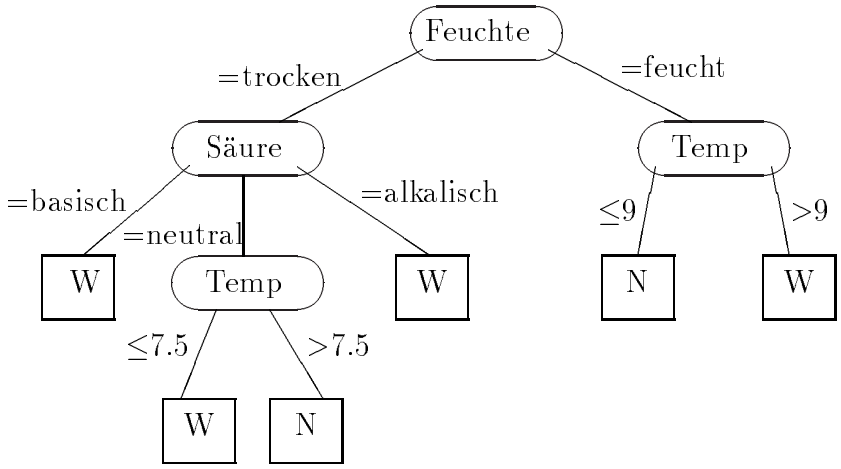

Abbildung 1.5: Gestutzter Entscheidungsbaum

gestutzten Baum zu erzeugen. In diesen Fällen, und dann, wenn eine separate Validierungsmenge zu aufwendig erscheint, kann man sich behelfen, indem man eine heuristische Schätzfunktion verwendet [Quinlan, 1993]. Diese Funktion kann beispielsweise die erzielte Vereinfachung des Baumes (z.B. Anzahl eingesparter Knoten) verrechnen mit dem durch diese Vereinfachung gestiegenen Trainingsfehler. Diese heuristische Steuerung des Stutzens kann recht gut funktionieren, generell ist jedoch eine separate Validierungsmenge vorzuziehen [Mingers, 1989].

1.3.2 Erweiterungen

In diesem Lehrbuchkapitel kann auf die Vielfalt mittlerweile entwickelter Entscheidungsbaumverfahren natürlich nicht eingegangen werden. Dennoch sollen kurz die wichtigsten Varianten skizziert werden.

Splitting-Kriterien Neben dem in diesem Kapitel vorgestellten Informationsgewinn-Kriterium sind viele andere Kriterien entwickelt worden. Einen Überblick gibt z.B. [Murthy, 1998].

Regelerzeugung Jeder Pfad von der Wurzel eines Entscheidungsbaumes bis zu einem Blattknoten entspricht einer logischen Regel. Vereinfacht man die einem Entscheidungsbaum entsprechende Regelmenge in geeigneter Weise, so entsteht ein leistungsfähiges Regellernverfahren [Quinlan, 1993].

Regression Schon relativ früh wurde erkannt, daß man in den Blattknoten eines Entscheidungsbaumes statt einer einfachen Klassenzuweisung auch eine lineare Regressionsfunktion angeben kann, um so auch numerische Werte vorhersagen zu können [Breiman *et al.*, 1984]. Mit dieser Funktion wird dann lokal im jeweiligen Blattknoten die Vorhersage für eine dorthin klassifizierte Instanz berechnet. Auch die Verwendung von linearen Trennebenen als Knotentests im Inneren des Baumes ist bereits untersucht worden [Murthy *et al.*, 1994].

Fehlende Werte Fehlende Werte in den Beispielen oder zu klassifizierenden Instanzen

sollten nicht durch die Einführung eines speziellen Sonderwertes wie "unknown" behandelt werden. Stattdessen ist es üblich, bei der Konstruktion des Baumes fehlende Werte gemäß der sonstigen Verteilung des betroffenen Attributes zu ergänzen. Bei der Klassifikation von Instanzen mit fehlenden Werten kann man entsprechend in probabilistischer Weise alle möglichen Pfade durch den Baum verfolgen und die Vorhersagen aller so erreichten Blattknoten gemäß ihrer Wahrscheinlichkeiten gewichtet miteinander kombinieren.

Skalierbarkeit Bei der Konstruktion eines Entscheidungsbaumes müssen die n Beispiele grob gesprochen auf ungefähr $log\ n$ Ebenen betrachtet werden, so daß der Aufwand mindestens in der Größenordnung von $n\ log\ n$ liegt. Damit gehören Entscheidungsbaumverfahren zwar zu den schnellsten Lernverfahren, dennoch muß man sich für größere Datenbestände auch hier Gedanken über die Skalierbarkeit machen. Standard ist hier die Verwendung eines *Lernfensters* ([Quinlan, 1993], *windowing*), in das anfänglich nur ein Bruchteil der Daten aufgenommen wird, und das dann wiederholt durch einen Bruchteil der vom jeweils erzeugten Entscheidungsbaum nicht korrekt klassifizierten anderen Trainingsdaten ergänzt wird, bis der Gesamtfehler sich nicht weiter verbessert. Auch existieren bereits Verfahrensvarianten, die durch geschickte Datenstrukturen in der Lage sind, den Aufwand für die wiederholte Sortierung der Daten in jedem Knoten zu vermeiden [Shafer *et al.*, 1996]. Diese Verfahren sind nicht darauf angewiesen, alle Daten im Hauptspeicher zu halten und konnen darüber hinaus leicht parallelisiert werden.

1.4 Instanzbasiertes Lernen

Unter der Bezeichnung "Instanzbasierte Lernverfahren" [Aha *et al.*, 1991] sind im Maschinellen Lernen Verfahren wiederaufgenommen und weiterentwickelt worden, die in der Statistik schon lange als "Nächste Nachbarn"-Verfahren [Cover und Hart, 1967] populär waren und sind. Instanzbasierte Verfahren beruhen auf einer bestechend einfachen Idee. Um die Klasse eines neuen, unbekannten Objektes vorherzusagen, vergleicht man das zu klassifizierende Objekt mit den gegebenen Beispielen, findet das Beispiel, das dem zu klassifizierenden Objekt am *ähnlichsten* ist, und sagt dann den für dieses Beispiel gespeicherten Funktionswert voraus.

Auch instanzbasierte Lernverfahren lösen also die Lernaufgabe "Funktionslernen aus Beispielen", aber auf ganz andere Art als die im vergangenen Abschnitt vorgestellten Entscheidungsbaumverfahren. Während Entscheidungsbaumverfahren die übergebene Beispielmenge E analysieren und auf ihrer Basis eine kompakte Hypothese in Form eines Entscheidungsbaumes produzieren, besteht die Lernphase bei instanzbasierten Lernverfahren einfach nur aus dem Abspeichern der Beispiele. Arbeit investieren diese Verfahren erst dann, wenn tatsächlich der Funktionswert für ein neues unbekanntes Objekt vorhergesagt werden soll. Man bezeichnet die instanzbasierten Lernverfahren deswegen oft auch als *lazy learners*.

Im Gegensatz zu Entscheidungsbäumen, die zur Klassifikation sehr schnell durchlaufen werden können, ist die Klassifikation bei instanzbasierten Verfahren aufwendiger, da das

neue Objekt mit allen bisherigen Objekten verglichen werden muss. Die große Popularität von instanzbasierten Lernverfahren erklärt sich zum einen dadurch, dass es für viele Nutzer natürlicher und überzeugender ist, als Begründung für eine getroffene Vorhersage nicht eine abstrakte Hypothese wie einen Entscheidungsbaum zu erhalten, sondern einfach Verweise auf ähnliche Instanzen aus den bekannten Beispielen E. Zum zweiten sind instanzbasierte Lernverfahren, trotz (bzw. gerade wegen) ihres einfachen Prinzips hervorragende Funktionsapproximatoren, deren Genauigkeit oft über der Genauigkeit von Entscheidungsbäumen oder anderen Lernverfahren liegt. Wie wir unten sehen werden, lassen sich darüber hinaus mit instanzbasierten Lernverfahren ebenso einfach diskrete Klassifikationen wie numerische Vorhersagen lernen.

Zur Präzisierung der von instanzbasierten Lernverfahren gelösten Lernaufgabe müssen wir zunächst überlegen, was eigentlich der Hypothesenraum eines solchen Verfahrens ist. Zur Erinnerung: Die Hypothese h ist eine Funktion, die eine Instanz $x \in X$ auf einen vorhergesagten Funktionswert $h(x)$ abbildet. Bei instanzbasierten Lernverfahren geschieht dies durch einen geeigneten Vergleichsalgorithmus auf der Basis der gegebenen Beispiele E. Da die Klassifikationsfunktion, wie unten beschrieben, meistens noch durch weitere Parameter gesteuert wird, definieren wir die Lernaufgabe für instanzbasierte Lernverfahren wie folgt.

Definition 1.4.1 (Instanzbasiertes Lernen) *Instanzbasierte Lernverfahren lösen die Lernaufgabe* Funktionslernen aus Beispielen*, wobei X eine Menge von durch n numerische oder diskrete Attribute beschriebenen Instanzen ist, Y eine numerische oder diskrete Menge von Zielwerten ist, E mit Beispielen $(x, y) \in X \times Y$ gegeben ist und $L_H := \{(D, P) \mid D \subseteq E, P \in \mathcal{P}\}$, wobei $\mathcal{P}$ die Menge aller zulässigen Parametrisierungen der instanzbasierten Klassifikationsfunktion ist.*

Diese Definition der Lernaufgabe erlaubt es also einem instanzbasierten Lernverfahren, etwas weniger "faul" zu sein. Das Verfahren kann sich entscheiden, nur eine Teilmenge D der gegebenen Beispiele E für die Klassifikation aufzuheben und es kann während der Lernphase bestimmte Parameterwerte P bestimmen, die später der Klassifikationsprozedur übergeben werden. Das oben beschriebene allereinfachste instanzbasierte Lernverfahren hat nach dieser Definition effektiv einen Hypothesenraum der Größe 1, denn es wählt immer alle Beispiele als Basis für die Klassifikation aus, und die Klassifikationsfunktion benötigt keine weiteren Parameter.

Wie sieht nun diese einfachste Klassifikationsfunktion aus? Nehmen wir zunächst an, die *Ähnlichkeit* zweier Instanzen $x_1 \in X, x_2 \in X$ sei durch eine Funktion

$$sim : X \times X \to [0, 1]$$

definiert, so dass höhere Werte von sim ähnlicheren Paaren von Instanzen entsprechen. Dann lässt sich die einfachste instanzbasierte Klassifikationsfunktion sehr kurz definieren:

$$h(x_{new}) := argmax(y)_{(x,y)\in E}\ sim(x, x_{new})$$

Es wird also der gespeicherte Funktionswert y desjenigen Beispiels (x, y) verwendet, das der zu klassifizierenden Instanz x_{new} am ähnlichsten ist[5].

[5] Bei Gleichstand wollen wir hier und im folgenden immer annehmen, dass zufällig ausgewählt wird.

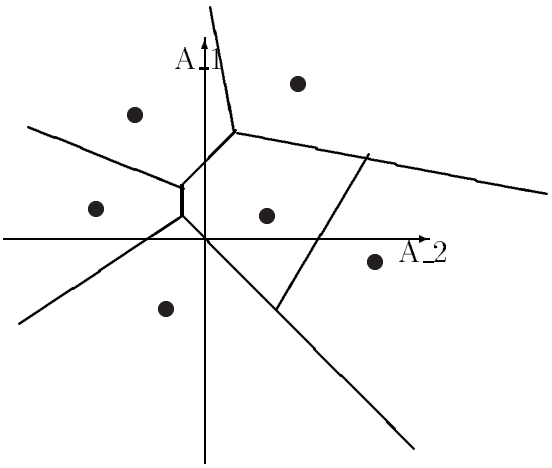

Abbildung 1.6: Voronoi-Diagramm für Nachbarschaftsbeziehungen

Dieses einfachste aller instanzbasierten Lernverfahren wird üblicherweise als *Nächste-Nachbarn*-Verfahren (NN, *nearest neighbor*) bezeichnet. Abbildung 1.6 visualisiert für Instanzen mit zwei reellwertigen Attributen grafisch die beim NN-Verfahren entstehende kleinteilige Parzellierung des Instanzenraums ("Voronoi - Diagramm"). Alle in eine Zelle dieses Diagramms fallenden Instanzen haben das eingezeichnete Beispiel als nächsten Nachbarn.

Diese kleinteilige Parzellierung ist für eine hohe Klassifikationsgüte zunächst einmal von Vorteil. Während verallgemeinernde Lernverfahren, wie z. B. die Entscheidungsbaumverfahren, mit ihren Hypothesen größere Bereiche des Instanzenraumes abdecken, bildet ein instanzbasiertes Lernverfahren sehr viele lokale Hypothesen, beim hier visualisierten NN-Verfahren tatsächlich eine lokale Hypothese pro Beispiel. Genau diese Möglichkeit zur lokalen Hypothesenbildung birgt aber auch ein großes Risiko der Überanpassung. Einzelne verrauschte oder sonstwie außergewöhnliche Datenpunkte beeinflussen direkt die Klassifikation, auch wenn sie keinen generelleren Zusammenhang wiedergeben.

Aus diesem Grund wird üblicherweise die k-NN (k-Nächste-Nachbarn) genannte Variante des Verfahrens verwendet, bei der die Vorhersage nicht aus einem nächsten Nachbarn, sondern aus den k nächsten Nachbarn gewonnen wird. Bei diskreten Vorhersageproblemen erfolgt dies naheliegenderweise durch eine Mehrheitsentscheidung innerhalb der Gruppe der k nächsten Nachbarn. Der Beitrag eines bestimmten Nachbarn zur Entscheidung wird dabei üblicherweise so gewichtet, dass er mit zunehmender Entfernung immer geringer wird.

Definition 1.4.2 (k-NN, diskret) *Gegeben eine Beispielmenge E, eine zu klassifizierende Instanz x_{new}, eine natürliche Zahl k und eine Ähnlichkeitsfunktion sim, bezeichne mit $N \subseteq E$ die k nächsten Nachbarn von x_{new}, und mit $N_{y_j} := \{e = (x, y_j) \in N\}$. die Nachbarn, die für den Wert y_j stimmen. Die k-NN Vorhersage ist dann definiert als:*

$$h(x_{new}) := argmax_{y_j} \sum_{e=(x,y_j) \in N_{y_j}} sim(x, x_{new})^a,$$

wobei a ein Parameter des Verfahrens ist.

Bei kontinuierlichen Vorhersageproblemen ist es weder notwendig noch sinnvoll, eine Mehrheitsentscheidung herbeizuführen. Stattdessen bildet man hier den — wiederum entsprechend der Ähnlichkeiten gewichteten — Mittelwert der gespeicherten Funktionswerte der nächsten Nachbarn.

Definition 1.4.3 (k-NN, kontinuierlich) *Seien E, x_{new}, k, N und sim wie bei der vorangegangenen Definition. Dann definiere:*

$$h(x_{new}) := \frac{\sum_{e=(x,y)\in N} sim(x, x_{new})^a \cdot y}{\sum_{e=(x,y)\in N} sim(x, x_{new})^a}$$

Die Parameter k und a könnten vom Benutzer vorgegeben werden. Üblich ist allerdings, dass a beim Entwurf des Verfahrens fest eingestellt wird (meistens auf den Wert 1 oder 2), und dass k vom Verfahren in der Lernphase bestimmt wird, (siehe unten), so dass k Teil der oben eingeführten Verfahrensparameter $\mathcal{P}$ wird.

1.4.1 Die Ähnlichkeitsfunktion

Wie kann man nun die Ähnlichkeit bzw. den Abstand zweier Instanzen berechnen? Falls die Instanzen ausschließlich durch numerische Attribute beschrieben sind, bietet sich die Verwendung des euklidischen Abstandes aus der Geometrie an. Da wir aber auch diskrete Attribute zulassen wollen, müssen wir die Abstandsfunktion etwas allgemeiner formulieren. Dies kann zum Beispiel dadurch geschehen, dass wir als Abstand den Durchschnitt der Abstände entlang der einzelnen Attribute verwenden.

Allgemein verlangt man für eine Abstandsfunktion *dist* die folgenden Eigenschaften.

1. $dist : X \times X \to \mathbb{R}^+, dist(x,x) = 0 \ \forall x \in X$,
2. $dist(x_1, x_2) = dist(x_2, x_1) \ \forall x_1, x_2 \in X$,
3. $dist(x,y) \leq dist(x,z) + dist(z,y) \ \forall x, y, z \in X$.

Sind die Bedingungen 1 bis 3 erfüllt, so ist *dist* eine *Metrik*. Bedingung 3 wird von den meisten Verfahren nicht ausgenutzt, so dass man meistens nur 1 und 2 fordert. Da es üblich ist, den Wertebereich der Funktion *dist* auf das Intervall $[0,1]$ zu skalieren, kann man statt einer Distanzfunktion auch eine Ähnlichkeitsfunktion verwenden:

$$dist(x,y) := 1 - sim(x,y) \ \forall x, y \in X$$

Wird *sim* wie folgt definiert, so ist das sich daraus ergebende Distanzmaß tatsächlich eine Metrik.

Definition 1.4.4 (Ähnlichkeitsfunktionen) *Seien x_1, x_2 Instanzen aus X, jeweils mit Attributen $A_1, \ldots, A_n$. Seien $w_1, \ldots, w_n$ nicht negative reelle Zahlen, die* Attributgewichte. *Dann definiere:*

$$sim(x_1, x_2) := \frac{\sum_{e=1,\ldots,n} w_i \cdot sim_{A_i}(x_1[i], x_2[i])}{\sum_{i=1,\ldots,n} w_i},$$

wobei $x[i]$ den Wert von Attribut A_i im Beispielvektor x bezeichnet.

Für ein numerisches Attribut A mit minimal (maximal) auftretenden Werten min_A (max_A)*, definiere weiter:*

$$sim_A(v_1, v_2) := 1 - \frac{\mid v_1 - v_2 \mid}{\max_A - min_A}$$

Für ein diskretes Attribut A, definiere:

$$sim_A(v_1, v_2) := 1, \textit{ falls } v_1 = v_2, \textit{ und 0 anderenfalls}^6.$$

Die Verwendung geeigneter Attributgewichte w_i ist in den meisten Fällen für den Erfolg des k-NN-Verfahrens ausschlaggebend. Dies liegt daran, dass dieses Verfahren, im Gegensatz z. B. zu Entscheidungsbaumverfahren, standardmäßig alle Attribute in die Abstandsberechnung einbezieht. Sind in den Instanzenbeschreibungen nun Attribute enthalten, die für die gesuchte Vorhersage überhaupt nicht notwendig sind, und daher auch in keiner systematischen Weise mit dem vorherzusagenden Wert korrelieren, so verfälschen die Attributswerte ohne die Verwendung von Gewichten das Abstandsmaß mehr oder weniger stark. Gewichte ermöglichen es nun, statt Attribute völlig aus den Instanzen zu verbannen, ihnen auch graduell eine geringere Bedeutung zu verleihen. Wo möglich sollten diese Attributgewichte vom Benutzer festgelegt werden, viele Verfahren können jedoch diese Attributgewichte auch als Teil ihrer Trainingsphase bestimmen. In diesem Fall werden also dann auch die w_i Teil der oben eingeführten Parameter $\mathcal{P}$ für das k-NN-Verfahren.

Allgemein sollte betont werden, dass die Wahl einer problemadäquaten Abstandsfunktion entscheidend für den Erfolg von instanzbasierten Lernverfahren ist. Wenn auch einfacher Euklidischer Abstand bzw. Funktionen wie die oben definierte üblich sind, so gibt es viele populäre Variationen, wie z.B. aus der Statistik die Mahalanobis-Abstände (siehe z.B. [Rencher, 1998]), welche auch die Kovarianzen der Variablen berücksichtigen.

1.4.2 Parameterbestimmung durch Kreuzvalidierung

Bezieht man Parameter wie k oder auch die Gewichte w_i in den Hypothesenraum L_H ein, so kann das k-NN-Verfahren seine Lernphase nicht mehr ganz so "faul" gestalten, denn nun muss es in der Lernphase diese Parameter ja bestimmen. Ähnlich wie beim Stutzen von Entscheidungsbäumen darf man beim Bestimmen dieser Parameter nun nicht einfach die resultierende Trainingsgenauigkeit zum Maßstab nehmen, denn dadurch droht wieder eine Überanpassung. Es kann z. B. eine unabhängige Evaluierungsmenge zur Schätzung des wahren Fehlers verwendet werden. Bei k-NN-Verfahren kommt dazu üblicherweise die Technik der *Kreuzvalidierung* zum Einsatz, die einen noch besseren Schätzer für den wahren Fehler darstellt als die Verwendung einer einzigen separaten Evaluierungsmenge[7].

Definition 1.4.5 (m-fache-Kreuzvalidierung) *Sei E eine Beispielmenge, und L ein Lernverfahren. Dann partitioniere E in m möglichst gleichgroße Teilmengen* $E_1 \ldots, E_m$*. Erzeuge nun die Hypothesen* $h_1, \ldots, h_m$ *wie folgt:*

[6] Alternativ kann hier auch eine benutzerdefinierte Einzelwert-Abstandsfunktion verwendet werden.

[7] Weitere Alternativen zur Schätzung des wahren Fehlers diskutiert [Scheffer und Joachims, 1999].

h_i := Ergebnis von L auf Basis der Beispielmenge $E \backslash E_i$.
Dann schätze den wahren Fehler von L auf E wie folgt:

$$error_{CV(E_1,\ldots,E_m)}(L,E) := \frac{\sum_{i=1,\ldots,m} error_{E_i}(h_i)}{m}$$

Es wird also bei der Kreuzvalidierung jede der m Teilmengen der Beispielmenge einmal als Evaluierungsmenge verwendet und dann der Durchschnitt aus den entstehenden Genauigkeiten gebildet. Das bei nur einer Evaluierungsmenge vorhandene Risiko, zufällig nur besonders leicht oder besonders schwer vorherzusagende Instanzen in der Evaluierungsmenge zu haben, wird dadurch vermieden, und es kommt zu einer besseren Schätzung, die um so genauer wird, je größer die Anzahl m der Teilmengen gewählt wird. Da für jede Teilmenge ein Lernlauf durchgeführt werden muss, wählt man bei Lernverfahren mit aufwendiger Lernphase entsprechend kleine Werte von m. Da bei einem einzelnen k-NN-Lauf ohne Parameterschätzung in der Lernphase nichts passiert, kann hier zur Schätzung des wahren Fehlers eine Kreuzvalidierung durchgeführt werden, bei der die Beispielmenge E in einelementige Teilmengen unterteilt wird. Es wird also jeweils genau eine Instanz aus der Trainingsmenge entfernt und als Testinstanz benutzt (*leave-one-out crossvalidation*).

Mit diesem Schätzer für den wahren Fehler kann man nun eine Suche nach den günstigsten Werten für die Parameter durchführen. Am einfachsten ist dies für den Parameter k: hier schätzt man einfach den jeweiligen Fehler, den die k-NN-Methode für einen bestimmten Wertebereich von k, beispielsweise von 1 bis 15 produziert, und wählt dann das günstigste k. Auch für die Attributgewichte kann eine Suche durchgeführt werden, z. B. mit der *hill-climbing* Suchtechnik (s. Kapitel). Üblicher ist es allerdings, die Attributgewichte zu bestimmen, indem man einen einfachen statistischen Test der Korrelation des jeweiligen Attributes mit den Zielwerten durchführt. Man gibt dann denjenigen Attributen, die die höchsten Korrelationen aufweisen, die höchsten Gewichte (einen Überblick gibt [Wettschereck *et al.*, 1997]).

REFERENZ AUF ANDERES BUCHKAPITEL

1.4.3 Weitere Verfahrensvarianten

Auch die grundlegende k-NN-Methode hat, ähnlich wie die Entscheidungsbäume, im Laufe der Zeit vielfältige Variationen erfahren, um bestimmte Eigenschaften der Methode weiter zu verbessern. Hier sind insbesondere die folgenden Gruppen von Ansätzen zu nennen.

Beispielauswahl. Wie ganz zu Anfang angemerkt, wird die Klassifikationsphase eines k-NN-Verfahrens dadurch verteuert, dass alle Beispiele betrachtet werden müssen. Tatsächlich kann man sich jedoch gut vorstellen, dass nicht alle Beispiele auch wirklich eine wichtige Rolle bei der Auswahl der nächsten Nachbarn für Testinstanzen spielen. Deshalb sind in verschiedenen Arbeiten (siehe z.B. [Aha *et al.*, 1991]) Vorschläge gemacht worden, wie man in der Lernphase eines k-NN-Verfahrens Instanzen identifizieren kann, die weitgehend ohne Schaden aus der Beispielmenge entfernt werden könnten. Diese Verfahren liefern dann also als Hypothese nur eine Teilmenge der Beispielmenge zurück.

Prototyp- und Rechteckansätze. Eine etwas weitergehende Idee stellen Prototyp- und Rechteckansätze dar (siehe z.B. [Salzberg, 1991; Wettschereck und Dietterich, 1995]).

Während bei der klassischen k-NN-Lernaufgabe, die wir oben definiert haben, nur die Auswahl von Beispielen erlaubt ist, gestattet man bei Prototypverfahren auch die Konstruktion neuer Instanzen, die als Durchschnitt oder Prototypen einer ganzen Gruppe von tatsächlich beobachteten Beispielen dienen, so dass diese Gruppe von Beispielen dann entfernt werden kann. Bei den Rechteckverfahren werden Gruppen von Beispielen nicht durch neukonstruierte Prototypbeispiele ersetzt, sondern durch eine allgemeine Beschreibung des entsprechenden Bereiches des Instanzenraums in Form eines Hyperrechtecks. Diese Rechtecke stellen natürlich eine echte Verallgemeinerung der gesehenen Beispiele dar, so dass diese Verfahren sich schon sehr deutlich von den instanzbasierten Lernverfahren wegbewegen in Richtung auf verallgemeinernde Lernverfahren wie die Entscheidungsbäume.

Mächtigere Instanzenräume. Bei einem instanzbasierten Lernverfahren — wenn es nicht Prototypen oder Rechtecke verwendet — wird auf den Instanzenraum X nur über die Ähnlichkeitsfunktion sim zugegriffen. Instanzbasierte Lernverfahren lassen sich also auch für mächtigere Instanzenräume verwenden, wenn es möglich ist, auf diesen Instanzenräumen entsprechende Ähnlichkeitsfunktionen zu definieren. Dies ist in verschiedenen Arbeiten beispielsweise für prädikatenlogische Instanzenräume durchgeführt worden (siehe z.B. [Emde und Wettschereck, 1996; Horvath *et al.*, 2000]).

1.5 Stützvektormethode

Die Stützvektormethode (engl. Support Vector Machine, SVM) [Vapnik, 1998] ist ein Lernverfahren, das direkt aus der statistischen Lerntheorie hervorgegangen ist. Im Sinne von SVMs beschreibt diese Theorie, wie gut aus einer gegebenen Menge von Beispielen auf weitere, ungesehene Beispiele geschlossen werden kann. Im einfachsten Fall behandeln auch SVMs die Lernaufgabe des Funktionslernens aus Beispielen[8]. In etwas verfeinerter Notation schreiben wir die Lernmenge E mit l Beispielen nun als $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_l, y_l)$. Jedes Beispiel besteht also aus einem Merkmalsvektor $\vec{x} \in X$ und einer Klassifikation $y \in Y$. Y enthält die Elemente $+1$ und -1. Für Instanzen des Begriffs ist $y = +1$ (positives Beispiel), für Nicht-Instanzen $y = -1$ (negatives Beispiel). Der Merkmalsvektor $\vec{x}^T = (x_1, \ldots, x_n)^T$ fasst in dieser Notation alle n Merkmalswerte des Beispiels zusammen. Anders als bei der "Standort"-Aufgabe in Abschnitt 1.3, müssen alle Merkmalswerte reelle Zahlen sein. $\vec{x}^T$ ist der transponierte Vektor zu $\vec{x}$.

SVMs werden besonders dann eingesetzt, wenn die Anzahl der Merkmale groß ist. Die Merkmale könnten z. B. die Bild-Pixel eines Fotos sein, wenn gelernt werden soll das Gesicht einer bestimmten Person zu erkennen. Bei einem Graustufenbild gibt dann der Merkmalswert x_i z. B. die Helligkeit des jeweiligen Pixels durch eine Zahl im Intervall [0..100] wieder. Positive Beispiele sind Fotos der entsprechenden Person, negative Beispiele sind Fotos anderer Personen (oder auch ganz anderer Objekte).

Oft werden SVMs auch zur Textklassifikation eingesetzt (z. B. [Joachims, 1998]). Hierunter fällt z. B. die Aufgabe, zwischen WWW-Seiten über Informatik und anderen Seiten zu unterscheiden. Die Merkmale sind hier nicht Bildpixel, sondern das Vorkommen von

[8] SVMs können auch zur Regression, Dichteschätzung, etc. benutzt werden (siehe [Vapnik, 1998]).

Wörtern auf der Seite. Für jedes Wort w der Sprache existiert ein zugehöriges Merkmal x_w. Der Wert des Merkmals x_w ist die Häufigkeit mit der das Wort w auf der WWW-Seite vorkommt. Positive Beispiele sind Seiten über Informatik, negative Beispiele sind beliebige andere Seiten.

Das Ziel von SVMs ist es nun aus den gegebenen Beispielen eine Klassifikationsregel abzuleiten, die neue Beispiele gleichen Typs mit größtmöglicher Genauigkeit klassifiziert. Anders gesagt möchte man die Wahrscheinlichkeit, auf einem neuen Beispiel die falsche Klasse vorherzusagen, minimieren. Aber wie erreichen SVMs dieses Ziel?

1.5.1 SVMs und die optimale Hyperebene

In ihrer einfachsten Form lernen SVMs lineare Klassifikationsregeln, d. h. L_H besteht aus Regeln h der Form

$$h(\vec{x}) = sign\left(b + \sum_{i=1}^{n} w_i x_i\right) = sign(\vec{w} \cdot \vec{x} + b)$$

Dabei sind $\vec{w}$ und b die Komponenten der Klassifikationsregel, die von der SVM angepasst werden. $\vec{w}$ ordnet jedem Merkmal ein gewisses Gewicht zu. Man nennt ihn deshalb Gewichtsvektor. Der Merkmalsvektor $\vec{x}$ des zu klassifizierenden Beispiels wird mit $\vec{w}$ durch das Skalarprodukt $\vec{w} \cdot \vec{x} = \sum_{i=1}^{n} w_i x_i$ verknüpft. b ist ein Schwellwert. $sign(a) = 1$, wenn a größer als null ist. Ansonsten ist $sign(a) = -1$. Wenn also der Wert von $\vec{w} \cdot \vec{x}$ plus den Schwellwert b größer als null ist, dann klassifiziert die Klassifikationsregel den Merkmalsvektor $\vec{x}^T = (x_1, \ldots, x_i, \ldots, x_n)^T$ als positives Beispiel, sonst als negatives Beispiel. In der einfachsten Form findet die SVM eine Klassifikationsregel, die alle Beispiele in der Lernmenge richtig klassifiziert. Diese Forderung kann man wie folgt formal aufschreiben:

$$y_1[\vec{w} \cdot \vec{x}_1 + b] > 0, \quad \ldots, \quad y_l[\vec{w} \cdot \vec{x}_l + b] > 0$$

Die SVM muss also den Gewichtsvektor $\vec{w}$ und den Schwellwert b so wählen, dass alle Ungleichungen erfüllt sind. Die SVM tut aber noch etwas mehr und findet eine sehr spezielle Belegung $(\vec{w}, b)$, die Vapnik die "optimale Hyperebene" nennt.

Eine Gleichung der Form $\vec{w} \cdot \vec{x} + b = 0$ beschreibt eine Hyperebene. Im zweidimensionalen Raum, wie in dem Beispiel in Abbildung 1.7, entspricht die Klassifikationsregel somit einer Geraden. Alle Merkmalsvektoren auf der einen Seite der Hyperebene werden als positive, alle Merkmalsvektoren auf der anderen Seite als negative Beispiele klassifiziert. Wie im linken Teil der Abbildung 1.7 ersichtlich, gibt es viele Geraden, die positive und negative Beispiele fehlerfrei trennen. Aus diesen wählt die SVM die Hyperebene aus, welche positive und negative Beispiele mit maximalem Abstand trennt. Dies ist die Hyperebene h^* im rechten Teil von Abbildung 1.7. Zu jedem Beispiel hat sie mindestens einen Abstand von δ. Formal kann man das Problem des Lernens bei SVMs durch das folgende Optimierungsproblem beschreiben:

$$finde \qquad \vec{w}, b \tag{1.1}$$

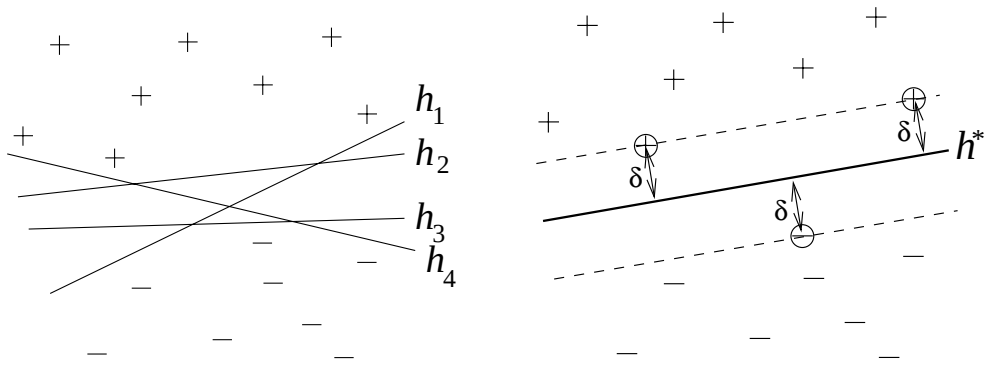

Abbildung 1.7: Ein binäres Klassifkationsproblem mit zwei Merkmalen. Positive Beispiele sind durch +, negative Beispiele durch − gekennzeichnet. Links sieht man, dass alle Hyperebenen h_1 bis h_4 die positiven von den negativen Beispielen trennen. Auf der rechten Seite ist die "optimale Hyperebene" h^* eingezeichnet, wie sie von der SVM gefunden wird. Sie trennt die Beispiele mit maximalem Abstand.

$$so\ dass \quad \delta \ \ maximal \tag{1.2}$$

$$und\ es\ gilt \quad y_1 \frac{1}{||\vec{w}||}\left[\vec{w}\cdot\vec{x}_1+b\right] \geq \delta, \ldots, y_l \frac{1}{||\vec{w}||}\left[\vec{w}\cdot\vec{x}_l+b\right] \geq \delta \tag{1.3}$$

$||\vec{w}|| \equiv \sqrt{\vec{w}\cdot\vec{w}}$ bezeichnet die Euklid'sche Länge des Vektors $\vec{w}$. In Worten lässt sich das Optimierungsproblem wie folgt zusammenfassen. Finde einen Gewichtsvektor $\vec{w}$ und einen Schwellwert b, so dass alle Beispiele der Lernmenge auf der "richtigen" Seite der Ebene liegen (d. h. die Ungleichungen (1.3) sind erfüllt) und der Abstand δ aller Beispiele von der Trennebene maximal ist. Es sei daran erinnert, dass der Absolutbetrag von $\frac{1}{||\vec{w}||}\left[\vec{w}\cdot\vec{x}+b\right]$ den Abstand des Punktes $\vec{x}$ von der Hyperebene berechnet. Die am nächsten an der Hyperebene liegenden Beispiele werden Stützvektoren (engl. Support Vectors) genannt. Ihr Abstand zur Hyperebene ist genau δ, d. h. die zugehörige Ungleichung ist per Gleichheit erfüllt. Die drei Stützvektoren sind in Abbildung 1.7 durch Kreise hervorgehoben. Eine interessante Eigenschaft von SVMs ist, dass allein die Stützvektoren die optimale Hyperebene bestimmen. Selbst wenn man alle anderen Beispiele wegließe, käme die SVM zu dem gleichen Ergebnis.

1.5.2 Wie berechnet man die optimale Hyperebene

Im vorangegangenen Abschnitt wurde beschrieben, wie die optimale Hyperebene definiert ist. Obwohl man recht einfach ihre formale Definition hinschreiben kann, wissen wir noch nicht, wie man sie mit einem Computer berechnet. Hierzu bringt man das Optimierungsproblem zunächst in eine einfachere Form. Da die Länge des Gewichtsvektors $||\vec{w}||$ die Lage der Hyperebene nicht beeinflusst, kann sie z. B. durch $\delta = \frac{1}{||\vec{w}||}$ fixiert werden. Man substituiert nun alle Vorkommen von δ. Anstelle nun $\frac{1}{||\vec{w}||}$ zu maximieren, kann man äquivalent hierzu $\vec{w}\cdot\vec{w}$ minimieren und kommt zu dem folgenden Optimierungsproblem.

$$finde \quad \vec{w}, b \tag{1.4}$$

$$so\ dass \quad w \cdot w \quad minimal \tag{1.5}$$

$$und\ es\ gilt \quad y_1[\vec{w} \cdot \vec{x}_1 + b] \geq 1, \ldots, y_l[\vec{w} \cdot \vec{x}_l + b] \geq 1 \tag{1.6}$$

Aber auch dieses Minimierungsproblem ist noch immer schlecht mit numerischen Methoden zu lösen. Deshalb formt man es weiter in das folgende, äquivalente Optimierungsproblem um (eine genaue Herleitung findet sich in [Vapnik, 1998], Seite 404):

$$finde \quad \alpha_1, \ldots, \alpha_l \tag{1.7}$$

$$so\ dass \quad -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j) \quad minimal \tag{1.8}$$

$$und\ es\ gilt \quad \sum_{i=1}^{l} y_i \alpha_i = 0 \ \ und \ \ \forall i : 0 \leq \alpha_i \tag{1.9}$$

Dieses Optimierungsproblem ist ein quadratisches Programm in Normalform. Es gilt, das Minimum bezüglich der Variablen $\alpha_1, \ldots, \alpha_l$ zu finden. Alle α_i müssen hierbei größer oder gleich null sein und die Gleichung in (1.9) muss erfüllt sein. Es existiert eine Reihe von Algorithmen, um Probleme dieser Art zu lösen. Allerdings sind sie oft sehr ineffizient für grosse l. Speziell für SVMs entwickelt wurden u. a. die Algorithmen in [Joachims, 1999],[Platt, 1999]. Empirisch haben diese Algorithmen meist ein Laufzeitverhalten von ca. n^2. Der in [Joachims, 1999] beschriebene Algorithmus ist auf dem WWW verfügbar[9].

Hat man die Lösung $\alpha_1^*, \ldots, \alpha_l^*$ des Optimierungsproblems berechnet, kann man die Klassifikationsregel wie folgt ablesen.

$$h(x) = sign\left(b + \sum_{i=0}^{l} \alpha_i^* y_i \vec{x}_i \cdot \vec{x}\right) \tag{1.10}$$

Der Gewichtsvektor ergibt sich also als Linearkombination der Beispiele aus der Lernmenge $\vec{w} = \sum_{i=0}^{l} \alpha_i^* y_i \vec{x}_i$. Hierbei haben nur die Stützvektoren ein $\alpha_i^* > 0$. Den Schwellwert b kann man mit Hilfe eines beliebigen Stützvektors $\vec{x}_i$ bestimmen $b = y_i - \vec{w} \cdot \vec{x}_i$.

1.5.3 Statistische Eigenschaften der optimalen Hyperebene

Wir wissen nun, wie die SVM die Hyperebene berechnet, die positive und negative Beispiele mit dem maximalen Abstand trennt. Aber warum ist dies sinnvoll? Die folgenden zwei Sätze [Vapnik, 1998] machen Aussagen über den wahren Fehler von SVMs. Sie beschreiben den Erwartungswert des wahren Fehlers $error_D$ einer SVM, die auf $l-1$ Beispielen trainiert wurde - kurz $E[error_D(h_{l-1}, f)]$. Dies ist der wahre Fehler den die SVM durchschnittlich erreicht, wenn sie mit Trainingsmengen E der Größe $l-1$ von der Verteilung D und der Funktion f trainiert wird.

[9] http://www-ai.cs.uni-dortmund.de/svm_light

Satz 1.5.1 *Der erwartete wahre Fehler* $E[error_D(h_{l-1}, f)]$ *von* h_{l-1}*, der Hyperebene* $sign(\vec{w} \cdot \vec{x} + b)$ *die* $l-1$ *Beispiele mit maximalem Abstand trennt, ist beschränkt durch*

$$E[error_D(h_{l-1}, f)] \leq \frac{E[\#SV_l]}{l},$$

wobei $E[\#SV_l]$ *die erwartete Anzahl von Stützvektoren ist, wenn man* l *Beispiele mit der optimalen Hyperebene trennt. Der Erwartungswert auf der linken Seite der Gleichung ist also über Lernmengen der Grösse* $l-1$*, der Erwartungswert auf der rechten Seite über Lernmengen der Grösse* l*.*

Beweis: Dieser Beweis benutzt die Leave-One-Out Methode aus Abschnitt 1.4.2. Mit der Leave-One-Out Methode kann man schätzen, wie gut ungesehene Beispiele klassifiziert werden. Der Leave-One-Out Fehler $error_{CV}(\text{SVM}, E)$ ist die Anzahl der fehlerhaft klassifizierten Beispiele geteilt durch l. Er ist eine Schätzung für den wahren Fehler, welche der Lerner nach $l-1$ Beispielen erreicht. Es ist aus der Statistik bekannt [Lunts und Brailovskiy, 1967], dass $error_{CV}(\text{SVM}, E)$ und der wahre Fehler nach dem Lernen auf $l-1$ Beispielen den gleichen Erwartungswert haben, also $E[error_D(h_{l-1}, f)] = E[error_{CV}(\text{SVM}, E)]$. Der entscheidende Schritt zum Beweis des Satzes ist die Einsicht, dass die SVM nur dann beim Leave-One-Out Test einen Fehler machen kann, wenn das zurückgehaltene Beispiel ein Stützvektor ist. Denn nur Stützvektoren beeinflussen die optimale Hyperebene. Daraus folgt $error_{CV}(\text{SVM}, E) \leq \frac{\#SV_l}{l}$ und somit $E[error_{CV}(\text{SVM}, E)] \leq \frac{E[\#SV_l]}{l}$, was den Satz beweist. ∎

Wenn wir zusätzlich den Schwellwert $b = 0$ fordern, so gilt auch der folgende Satz. Er verbindet den erwarteten wahren Fehler mit der erwarteten Separationsweite.

Satz 1.5.2 *Der erwartete wahre Fehler* $E[error_D(h_{l-1}, f)]$ *von* h_{l-1}*, der Hyperebene* $sign(\vec{w} \cdot \vec{x} + 0)$ *die* $l-1$ *Beispiele mit maximalem Abstand trennt, ist beschränkt durch*

$$E[error_D(h_{l-1}, f)] \leq \frac{E[\frac{R^2}{\delta^2}]}{l},$$

wobei δ *die Separationsweite und* R *die Euklid'sche Länge des längsten Merkmalsvektors in der Trainingsmenge ist. Der Erwartungswert auf der linken Seite der Gleichung ist also über Lernmengen der Grösse* $l-1$*, der Erwartungswert auf der rechten Seite über Lernmengen der Grösse* l*.*

Besonders bemerkenswert ist, dass in keine der obigen Formeln die Anzahl der Merkmale eingeht. Im Gegensatz zu den anderen hier vorgestellten Lernverfahren können SVMs somit auch bei vielen Merkmalen gut lernen, solange die Aufgabe eine große Separationsweite oder wenige Stützvektoren aufweist. Für ein weiteres Argument zur Generalisierungsgenauigkeit von SVMs sei auf den Abschnitt zum PAC-Lernen verwiesen.

1.5.4 Nicht-lineare SVMs durch Kernfunktionen

Bis jetzt konnten wir mit SVMs lediglich lineare Klassifikationsregeln lernen. Für viele Probleme ist das allerdings nicht ausreichend. Sie haben eine nicht-lineare Struktur. Eine herausragende Eigenschaft von SVMs ist, dass man sie sehr einfach in nicht-lineare

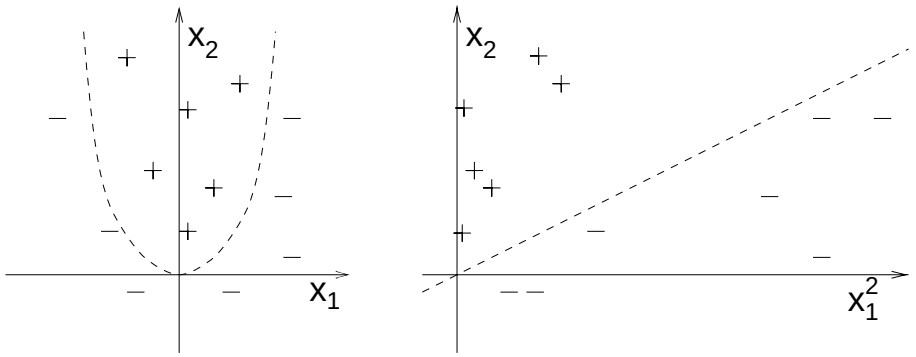

Abbildung 1.8: Die linke Graphik zeigt eine linear nicht trennbare Lernmenge im Eingaberaum (x_1, x_2). Die rechte Graphik zeigt das gleiche Problem nach der nichtlinearen Transformation, projiziert auf (x_1^2, x_2). Im diesem Raum sind die Beispiele linear trennbar.

Klassifikatoren umwandeln kann. Dies geschieht nach dem folgenden Prinzip. Die Merkmalsvektoren $\vec{x}_i$ werden durch eine nichtlineare Funktion $\Phi(\vec{x}_i)$ in einen hochdimensionalen Raum X' abgebildet. In X' lernt die SVM dann eine Hyperebene. Obwohl die trennende Klassifikationsregel in X' linear ist, ergibt sich im Eingaberaum eine beliebig komplexe, nichtlineare Klassifikationsregel. Hierzu folgt ein Beispiel mit zwei Merkmalen x_1, x_2. Als nichtlineare Abbildung wählen wir $\Phi((x_1, x_2)^T) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)^T$. Obwohl man im Eingaberaum die Beispiele im linken Bild von Abbildung 1.8 nicht linear trennen kann, sind sie nach der Transformation durch $\Phi(\vec{x})$ (rechts) linear trennbar. Dies leistet z. B. der Gewichtsvektor $(-1, 0, 0, 0, \sqrt{2}, 0)$ wie eingezeichnet.

Im Allgemeinen ist eine solche Abbildung $\Phi(\vec{x})$ ineffizient zu berechnen. Boser et al. [Boser *et al.*, 1992] haben jedoch eine besondere Eigenschaft von SVMs erkannt. Sowohl in der Lernphase, als auch bei der Anwendung der gelernten Klassifikationsregel braucht man lediglich Skalarprodukte in x' zu berechnen, also $\Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$. Dies kann für bestimmte Abbildungen $\Phi(\vec{x})$ sehr effizient mit einer sog. Kernfunktion $K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$ geschehen. Mit der Kernfunktion

$$K_{poly}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d$$

kann die SVM so Polynome vom Grad d lernen, mit

$$K_{rbf}(\vec{x}_i, \vec{x}_j) = exp(\gamma(\vec{x}_i - \vec{x}_j)^2)$$

sog. Radial Basis Klassifikatoren, und mit

$$K_{sigmoid}(\vec{x}_i, \vec{x}_j) = tanh(s(\vec{x}_i \cdot \vec{x}_j) + c)$$

zweilagige neuronale Netze. Der Einsatz von Kernfunktionen macht SVMs zu einer sehr flexiblen Lernmethode. Für das obige Beispiel ist $K_{poly}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^2$ die zugehörige Kernfunktion. Man braucht also die Abbildung $\Phi(\vec{x})$ nicht explizit zu berechnen.

In Formel (1.8) kann nun das Skalarprodukt $\vec{x}_i \cdot \vec{x}_j$ einfach durch den Wert der gewünschten Kernfunktion $K(\vec{x}_i, \vec{x}_j)$ ersetzt werden. Hat man das Optimierungsproblem gelöst, kann

man ebenso in der Klassifikationsregel (1.10) das Skalarprodukt $\vec{x}_i \cdot \vec{x}$ durch $K(\vec{x}_i, \vec{x})$ ersetzen und erhält:

$$h(x) = sign\left(b + \sum_{i=0}^{l} \alpha_i^* y_i K(\vec{x}_i, \vec{x})\right)$$

1.5.5 SVMs mit "weicher" Trennung

Viele reale Lernmengen lassen sich auch mit "komplexen" Klassifikationsregeln nicht fehlerfrei trennen. Um Trainingsfehler beim Lernen zulassen zu können, benutzt man SVMs mit weicher Trennung [Cortes und Vapnik, 1995].

$$finde \quad \vec{w}, b \tag{1.11}$$

$$so\ dass \quad w \cdot w + C \sum_{i=1}^{l} \xi_i \quad minimal \tag{1.12}$$

$$und\ es\ gilt \quad y_1[\vec{w} \cdot \vec{x_1} + b] \geq 1 - \xi_1, \ldots, y_l[\vec{w} \cdot \vec{x_l} + b] \geq 1 - \xi_l \tag{1.13}$$

Man minimiert nicht wie zuvor allein die Länge des Gewichtsvektors, sondern zusätzlich die Summe der Fehler ξ_i. Jedes ξ_i misst, wie weit das entsprechende Beispiel im oder jenseits des Separationsbereichs liegt. C ist ein Parameter, der ein Abwägen von Separationsweite und Fehler erlaubt. Das duale quadratische Optimierungsproblem in Standardform für SVMs mit "weicher" Trennung gleicht dem für SVMs mit "harter" Trennung, wenn man in Gleichung (1.9) $\forall i : 0 \leq \alpha_i$ durch $\forall i : 0 \leq \alpha_i \leq C$ ersetzt.

Als weiterführende Literatur ist das Tutorium von Burges [Burges, 1998] und das Buch [Christianini und Shawe-Taylor, 2000] zu empfehlen. Das Standardwerk zu SVMs ist [Vapnik, 1998].

1.6 Lernbarkeit in wahrscheinlich annähernd korrektem Lernen (PAC)

Dieses Kapitel beschäftigt sich mit der Frage, welche Funktionsklassen überhaupt lernbar sind. Es definiert zum einen was es heisst, einen Begriff zu lernen. Zum anderen liefert es Kriterien dafür, welche Aufgaben nicht lernbar sind. Auch wenn die Verbindung von Theorie und Praxis oft nur eingeschränkt möglich ist, sind diese theoretischen Aussagen auch für reale Anwendungen relevant. Ein beweisbar nicht lernbares Problem sollte man nicht angehen, es sei denn man kann zusätzliche Einschränkungen treffen.

Seit Leslie Valiants Artikel hat sich die komplexitätstheoretische Herangehensweise an das Problem der Lernbarkeit von Begriffen aus Beispielen durchgesetzt [Valiant, 1984]. Das von ihm eingeführte Paradigma des wahrscheinlich annähernd korrekten Lernens (*probably approximately correct learning* – **PAC-learning**) geht von der Überlegung aus, dass es völlig aussichtslos ist, ein korrektes und vollständiges Lernverfahren zu fordern, das nach einer bestimmten Menge von Eingaben sicher und prompt das richtige Ergebnis abliefert

und dann anhält. Beim *PAC-learning* schwächt man die Anforderung an die Korrektheit des Lernergebnisses ab. Das Lernergebnis ist nur noch mit einer bestimmten Wahrscheinlichkeit von $1-\delta$ mit einem Fehler von höchstens ϵ richtig. Es wird also nur approximiert, nicht mehr identifiziert. Der Abschwächung bei der Korrektheit stehen aber zwei schwierige Anforderung an das Lernen gegenüber: Das Lernen soll in polynomiell beschränkter Rechenzeit zum Ergebnis kommen, nachdem es eine beschränkte Zahl von Beispielen gesehen hat. Die Beispiele werden hierbei nach einer unbekannten aber festen Wahrscheinlichkeitsverteilung gezogen. Die Lernmenge ist also eine Stichprobe. Das Lernergebnis soll die Begriffsdefinition oder Erkennungsfunktion für den Begriff sein.

Hier wird das Szenario des *PAC-learning* kurz vorgestellt. Eine kurze, übersichtliche Einführung bietet [Hoffmann, 1991], eine ausführliche Behandlung des Bereiches bieten [Kearns und Vazirani, 1994], [Kearns, 1990], [Anthony und Biggs, 1992], [Natarajan, 1991].

Für eine gegebene Menge X von Instanzenbeschreibungen (alle möglichen Beispiele) ist ein Begriff c eine Teilmenge von X (nämlich alle Objekte, die der Begriff abdeckt) und eine Begriffsklasse C eine Teilmenge der Potenzmenge von X. Meist wird C durch die Form der Begriffe spezifiziert (z.B. Boolsche Funktionen oder Formeln in einer Normalform mit k Termen). Oft setzt man den Begriff mit der Klassifikation durch den Begriff gleich. Wie beim Funktionslernen ist c dann eine Funktion, die X auf 1 (positives Beispiel) oder 0 (negatives Beispiel) abbildet. Analog zur Repräsentationsklasse C des Zielbegriffs c wird die Repräsentationsklasse H des Lernergebnisses h definiert. H ist der Hypothesenraum, h eine Hypothese. Wenn $h(x) = c(x)$ für alle x einer Lernmenge E ist, so sagt man die Hypothese h ist konsistent mit den gegebenen Beispielen.

Definition 1.6.1 (PAC-Lernbarkeit) *Eine Begriffsklasse C mit Instanzen x der Größe $\mid x \mid$ ist (PAC-)lernbar durch einen Hypothesenraum H, wenn es einen Algorithmus L gibt, der*

- *für alle $0 \leq \epsilon \leq 1/2$ und $0 \leq \delta \leq 1/2$*
- *bei allen beliebigen aber festen Wahrscheinlichkeitsverteilungen D über X*
- *für alle $c \in C$*
- *in einer Zeit, die durch ein Polynom über $1/\epsilon, 1/\delta, \mid c \mid, \mid x \mid$ begrenzt ist*
- *mit einer Wahrscheinlichkeit von mindestens $1-\delta$*

eine Hypothese $h \in H$ ausgibt, deren wahrer Fehler $error_D(h,c)$ nicht größer ist als ϵ.

Die Repräsentationsgröße $\mid c \mid$ wird unterschiedlich angegeben. Es kann z.B. die Länge einer Repräsentation bei effizienter Codierung sein. Die Forderung der polynomiellen Laufzeit impliziert, dass auch die Anzahl der Beispiele polynomiell beschränkt ist.

In diesem Szenario kann man nun für bekannte Sprachklassen bzw. ihre Automaten die prinzipielle Lernbarkeit von Begriffen, die in dieser Sprache ausgedrückt sind, untersuchen. In den letzten Jahren ist eine Fülle von Beweisen zur Lernbarkeit bestimmter Repräsentationsklassen erarbeitet worden. Interessanterweise ist die Hierarchie der Lernbarkeit nicht dieselbe wie die übliche Komplexitätshierarchie. Das bekannte Beispiel hierfür sind k-KNF und k-Term-DNF.

Definition 1.6.2 (k-KNF) *Eine Formel $C_1 \wedge ... \wedge C_l$, bei der jedes C_i eine Disjunktion von höchstens k Literalen über Boolschen Variablen ist, ist in k-konjunktiver Normalform*

(k-KNF).

Definition 1.6.3 (k-Term-DNF) *Eine Formel $T_1 \vee ... \vee T_k$, bei der jeder Term T_i eine Konjunktion über höchstens n Boolschen Variablen ist, ist in k-Term disjunktiver Normalform (k-Term-DNF).*

Natürlich ist k-KNF ausdrucksstärker als k-Term-DNF, denn man kann jeden Ausdruck in k-Term-DNF in k-KNF schreiben, aber nicht umgekehrt. Die Repräsentationsklasse $C = k{-}KNF$ ist lernbar durch $H = k{-}KNF$. Die Repräsentationsklasse $C = k{-}Term{-}DNF$ ist nicht lernbar durch $H = k - Term - DNF$, wohl aber lernbar durch $H = k - KNF$. Der mächtigere Hypothesenraum, der immer noch polynomiell lernbar ist, erleichtert das Lernen. Der 'kleinere' Hypothesenraum macht das Lernen schwieriger, weil je Beispiel mehr Rechenzeit benötigt wird. Die Lernbarkeit ergibt sich ja einerseits aus der Anzahl der benötigten Beispiele, andererseits aus der Rechenzeit pro Beispiel. Obwohl das Lernen eines Begriffs in der Klasse der k-DNF-Formeln nur polynomiell viele Beispiele braucht, ist er nicht lernbar, weil er im schlimmsten Falle zu lange für die Verarbeitung eines Beispiels braucht. Man kann die Anzahl der Beispiele abschätzen, die man dem Algorithmus geben muss, damit er lernen kann (Stichprobenkomplexität). Damit hat man dann den ersten Schritt eines PAC-Lernbarkeitsbeweises geschafft.

1.6.1 Stichprobenkomplexität

Die Stichprobenkomplexität gibt an, nach wie vielen Beispielen der Lerner einen Begriff spätestens gelernt hat.

Definition 1.6.4 (Stichprobenkomplexität) *Die Stichprobenkomplexität eines Algorithmus für gegebenes H, C, ϵ und δ ist die Anzahl von Beispielen l, nach denen der Algorithmus spätestens ein beliebiges $c \in C$ mit Wahrscheinlichkeit von mindestens $1 - \delta$ bis auf einen Fehler von höchstens ϵ approximiert hat.*

Nehmen wir an, ein Lernalgorithmus gibt nur die Hypothesen aus, die alle positiven Beispiele und kein negatives Beispiel abdecken. Wenn wir weiterhin annehmen, dass der Zielbegriff in der Hypothesensprache enthalten ist, dann können wir recht einfach die Stichprobenkomplexität berechnen. Zunächst betrachten wir Hypothesenräume mit endlicher Größe $\mid H \mid$.

Satz 1.6.1 (Stichprobenkomplexität endlicher Hypothesenräume) *Für endliche Hypothesenräume H mit $H = C$ kann spätestens nach*

$$l \geq \frac{1}{\epsilon}(ln \mid H \mid + ln(1/\delta)) \tag{1.14}$$

Beispielen jedes $c \in C$ mit Wahrscheinlichkeit von mindestens $1 - \delta$ bis auf einen wahren Fehler von höchstens ϵ gelernt werden.

Beweis: David Haussler hat die Formel 1.14 anhand des Versionenraums bewiesen [Haussler, 1988] (ähnliche Schranken wurden zuvor auch von Vapnik bewiesen [Vapnik, 1998]). Der Versionenraum enthält ja zu jedem Zeitpunkt nur mit den bisher gesehenen Beispielen konsistente Hypothesen und der Hypothesenraum ist endlich. Damit erfüllt er

genau die Annahmen. Der Beweis betrachtet den schlimmsten Fall: bezüglich der bisher gesehenen Beispiele ist eine Hypothese im Versionenraum zwar korrekt, tatsächlich ist ihr Fehler aber größer als ϵ. Die Wahrscheinlichkeit, dass eine beliebige Hypothese mit Fehler größer als ϵ ein zufällig gezogenes Beispiel richtig klassifiziert, ist also höchstens $1-\epsilon$. Bei l Beispielen beträgt die Wahrscheinlichkeit also höchstens $(1-\epsilon)^l$. Bei k Hypothesen ist die Wahrscheinlichkeit, dass eine von ihnen schlecht ist (d. h. einen Fehler größer als ϵ hat), höchstens $k(1-\epsilon)^l$. k kann natürlich nicht größer sein als der Hypothesenraum. Im schlimmsten Fall ist die Wahrscheinlichkeit, dass eine der Hypothesen im Versionenraum einen Fehler größer als ϵ hat, also $\mid H \mid (1-\epsilon)^l$. Dies beschränkt die Wahrscheinlichkeit, dass l Beispiele nicht ausreichten, um die schlechten Hypothesen aus dem Versionenraum zu tilgen. Da ϵ zwischen 0 und 1 liegt, können wir sie abschätzen als $\mid H \mid e^{-\epsilon l}$ (oft additive Chernoff-Schranke genannt). Damit ein Begriff PAC-gelernt wird, muss diese Wahrscheinlichkeit kleiner sein als δ:

$$\mid H \mid e^{-\epsilon l} \leq \delta$$

Eine Umformung ergibt genau 1.14. ∎

Bei unendlichen Hypothesenräumen, wie z. B. den Hyperebenen der SVM, kann man die Größe nicht mehr durch einfaches Abzählen angeben. Allerdings gibt es auch in unendlichen Hypothesenraümen oft nur endlich viele Hypothesen, die sich wirklich substantiell unterscheiden. Die Ausdrucksstärke auch unendlicher Hypothesenräume gibt die **Vapnik-Chervonenkis-Dimension** (VC-Dimension) an. Sie misst die Kapazität ("Größe") eines Hypothesenraums und kann auch bei unendlichen Hypothesenräumen verwendet werden, um die Stichprobenkomplexität zu berechnen. Die VC-Dimension gibt an, wie viele Beispiele man maximal mit Hypothesen aus H beliebig aufteilen (d. h. zerschmettern) kann.

Definition 1.6.5 (Zerschmettern einer Beispielmenge) *Sei H der Hypothesenraum über X und S eine m-elementige Teilmenge von X. S wird von H zerschmettert (shattered), falls es für alle $S' \subseteq S$ eine Hypothese $h_{S'} \in H$ gibt, die genau S' abdeckt, d.h. $S \cap h_{S'} = S'$.*

Alle Teilmengen von S werden also durch Hypothesen in H erkannt.

Definition 1.6.6 (Vapnik-Chervonenkis-Dimension) *Die Vapnik-Chervonenkis-Dimension von H, $VCdim(H)$, ist die Anzahl der Elemente der größten Menge $S \subseteq X$, die von H zerschmettert wird.*

$$VCdim(H) = max\{m : \exists S \subseteq X, \mid S \mid = m, H \text{ zerschmettert } S\}$$

Wenn es kein Maximum der Kardinalität von S gibt, ist $VCdim$ unendlich.

Satz 1.6.2 (VC-Dimension endlicher Hypothesenräume) *Wenn der Hypothesenraum H endlich ist, dann ist $VCdim(H) \leq log_2(\mid H \mid)$.*

Beweis: Um eine Menge der Größe m zu zerschmettern, sind mindestens 2^m verschiedene Hypothesen nötig, weil es ja 2^m verschiedene Teilmengen gibt. ∎

Wie bestimmt man die VC-Dimension eines unendlichen Hypothesenraumes? Das folgende Beispiel untersucht die VC-Dimension von Hyperebenen, wie sie schon bei der SVM eingeführt wurden.

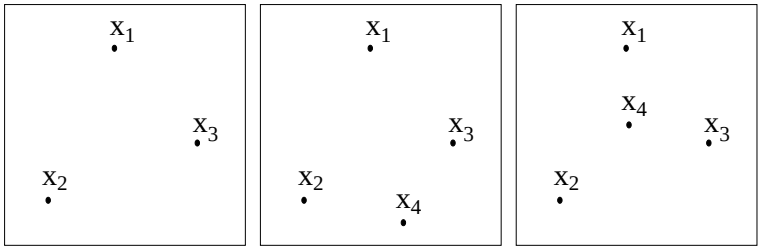

Abbildung 1.9: 3 und 4-elementige Teilmengen von X

- X: Punkte in einer Ebene, dargestellt durch $\vec{x} = (x_1, x_2)$;
- H: Hyperebenen $w_1x_1+w_2x_2+b$ mit zwei Gewichten w_1 und w_2 und einem Schwellwert b. Jeder Hyperebene entspricht die Boolsche Funktion $h(\vec{x}) = 1$ gdw. $w_1x_1+w_2x_2+b > 0$.

Für eine 3-elementige Teilmenge S von X gibt es 2^3 Möglichkeiten, die Elemente von S in positive und negative Beispiele zu klassifizieren (d. h. zu zerschmettern). Die 8 verschiedenen Aufteilungen sind:

S	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
(x_1, y_1)	-	+	-	+	-	+	-	+
(x_2, y_2)	-	-	+	+	-	-	+	+
(x_3, y_3)	-	-	-	-	+	+	+	+

Für die drei Beispielvektoren $\vec{x}_1$, $\vec{x}_2$ und $\vec{x}_3$ im linken Bild von Abbildung 1.9 können alle 8 Aufteilung durch Hyperebenen realisiert werden. Wie bei der SVM kann man sich eine Hypothese aus H als trennende Linie zwischen positiven und negativen Beispielen vorstellen. $VCdim(H)$ ist also mindestens 3. Aber könnte H nicht auch eine 4-elementige Teilmenge von X zerschmettern? Abbildung 1.9 zeigt, wie man 4 Punkte in der Ebene entweder konvex (Abb. 1.9, Mitte) oder nicht konvex (Abb. 1.9, rechts) anordnen kann. Im ersten Fall können $\{\vec{x}_1, \vec{x}_4\}$ und $\{\vec{x}_2, \vec{x}_3\}$ nicht durch eine Linie getrennt werden. Ebenso lässt sich im zweiten Fall $\{\vec{x}_4\}$ nicht von den anderen drei Beispielen linear trennen. Dies illustriert, dass die $VCdim(H)$ nicht nur mindestens 3, sondern genau 3 ist.

Im Allgemeinen gilt, dass die VC-Dimension von Hyperebenen im n-dimensionalen Raum gleich $n + 1$ ist. Man kann aber in vielen Fällen eine wesentlich kleinere $VCdim$ erhalten, wenn man die Separationsweite δ (siehe Abschnitt über SVMs) betrachtet.

Satz 1.6.3 *[Vapnik, 1998] Für Hyperebenen $h(\vec{x}) = sign(\vec{w} \cdot \vec{x} + b)$ im n-dimensionalen Raum, die l Beispiele wie folgt trennen,*

$$\forall_{i=0}^{l} : \frac{1}{||\vec{w}||}|\vec{w} \cdot \vec{x}_i + b| \geq \delta \quad und \quad ||\vec{x}_i|| \leq R$$

gilt, dass ihre $VCdim$ nicht größer ist als

$$min\left(\frac{R^2}{\delta^2}, n\right) + 1$$

Person	Partner	Mutter	Oma mütterlich
Doris	Dieter	Christa	Berta
Dieter	Doris	Carla	Bettina
Christa	Christoph	Berta	Andrea
Carla	Carlos	Bettina	Angela

Tabelle 1.4: Beispiele für den Begriff der Großmutter mütterlicherseits

In vielen Fällen ist es schwierig, die $VCdim$ genau zu bestimmen. Oft werden nur Abschätzungen gefunden. Den Zusammenhang zwischen der VC-Dimension einer Begriffsklasse und ihrer Stichprobenkomplexität geben die folgenden Ergebnisse an:

Satz 1.6.4 *$C = H$ hat polynomielle Stichprobenkomplexität gdw. $VCdim(C)$ endlich ist.*

Analog zur Hypothesenanzahl in endlichen Hypothesenräumen (Gleichung 1.14) kann man mit der VC-Dimension die Stichprobenkomplexität auch bei unendlichen Hypothesenräumen abschätzen.

Satz 1.6.5 (Stichprobenkomplexität unendlicher Hypothesenräume)
Für unendliche Hypothesenräume H mit $H = C$ kann spätestens nach

$$l \geq \frac{1}{\epsilon}\left(4log_2\left(\frac{2}{\delta}\right) + 8VCdim(H)log_2\left(\frac{13}{\epsilon}\right)\right)$$

Beispielen jedes $c \in C$ mit Wahrscheinlichkeit von mindestens $1 - \delta$ bis auf einen wahren Fehler von höchstens ϵ gelernt werden.

Wir können somit angeben, welche Anzahl von Beispielen ausreicht, um einen Begriff zu lernen. Allerdings ist die so berechnete Stichprobenkomplexität oft unrealistisch hoch. Meist reichen weit weniger Beispiele für ein gutes Lernergebnis aus. Desweiteren muss man sehr genau prüfen, ob die im PAC-Lernen gemachten Annahmen in der Anwendung wirklich zutreffen.

1.7 Begriffslernen mit induktiver logischer Programmierung

Bisher haben wir Begriffe gelernt, die durch Attribute beschrieben wurden. Bei einigen Begriffen ist das eine eher umständliche Repräsentation. Wie soll zum Beispiel der Begriff der *Ehe* oder der *Grossmutter* durch Attribute definiert werden? Wir können natürlich bei einem Menschen in Form von Attributen angeben, wer der Ehepartner ist und wer seine Großmütter sind. Beschränken wir uns auf den Begriff der Großmutter mütterlicherseits, so zeigt Tabelle 1.4 zwei Beispiele für diesen Begriff: Berta ist die Großmutter von Doris und Bettina die von Dieter. Wir können also die Beispiele in Form von Attributen aufschreiben. Aber wie können wir den allgemeinen Begriff darstellen? Dies gelingt mit Relationen sehr einfach. In Prädikatenlogik können wir Relationen und ihre Verknüpfungen darstellen. Wir

schreiben die Beispiele als zweistellige Prädikate:

$$\begin{array}{lll} verheiratet(doris, dieter) & mutter(doris, christa) & mutter(christa, berta) \\ verheiratet(christa, christoph) & mutter(dieter, carla) & mutter(carla, bettina) \\ verheiratet(carla, carlos) & & \end{array}$$

Den allgemeinen Begriff können wir als prädikatenlogische Formel schreiben:

$$mutter(X, Y) \wedge mutter(Y, Z) \rightarrow oma(X, Z)$$

Verwandschaftsbeziehungen sind relationale Begriffe. Andere typische Beispiele sind zeitliche und räumliche Beziehungen. Wenn wir von konkreten Zeitpunkten abstrahieren wollen, um Wissen für die Planung zu lernen, brauchen wir Relationen zwischen Zeitintervallen. Nehmen wir an, wir wollten einen allgemeinen Plan für die Handlungen lernen, die man mit dem Wort "abfliegen" zusammenfasst. So ein Plan könnte etwa folgendermaßen aussehen:

$$\begin{array}{rcl} zumFlughafen(Person, Flugh, T_1, T_2) & \wedge & \\ einchecken(Person, T_2, T_3) & \wedge & \\ gateFinden(Person, T_3, T_4) & \wedge & \\ einsteigen(Person, T_4, T_5) & \wedge & \\ warten(Person, T_5, T_6) & \rightarrow & abfliegen(Person, Flugh, T_6) \end{array}$$

$$\begin{array}{rcl} ort(Person, W) & \wedge & \\ fahren(Person, W, Flugh, T_1, T_2) & \rightarrow & zumFlughafen(Person, Flugh, T_1, T_2) \end{array}$$

$$\begin{array}{rcl} gehen(Person, W, Station, T_1, T_2) & \wedge & \\ warten(Person, T_2, T_3) & \wedge & \\ bahnFahren(Person, Station, Ziel, T_3, T_4) & \wedge & \\ gehen(Person, Ziel, Flugh, T_4, T_5) & \rightarrow & fahren(Person, W, Flugh, T_1, T_5) \end{array}$$

Beispiele für die allgemeinen Regeln enthalten natürlich konkrete Zeitpunkte.

$$\begin{array}{ll} zumFlughafen(uta, ddorf, 12:00, 13:36) & einchecken(uta, 13:36, 13:45) \\ gateFinden(uta, 13:45, 13:50) & einsteigen(uta, 13:50, 13:55) \\ warten(uta, 13:55, 14:06) & abfliegen(uta, ddorf, 14:06) \end{array}$$

Bei den verschiedenen Ereignissen des Abfliegens kommen jeweils verschiedene Zeiten vor, gemeinsam ist allen Ereignissen nur die Relation der Zeitintervalle. In diesem Beispiel wird durch die Konklusion ein umfassendes Zeitintervall ausgedrückt, die Prämissen drücken eine direkte Folge von Handlungen aus, d.h. das Ende der einen Handlung ist gerade der Beginn der nächsten Handlung.

Um relationale Begriffe ausdrücken zu kennen, brauchen wir eine eingeschränkte Prädikatenlogik. Dies kann z.B. eine terminologische Logik (Beschreibungslogik) sein oder ein logisches Programm. Das Erlernen logischer Programme, genannt **induktive logische Programmierung**, ist theoretisch gründlich untersucht worden (s. Abschnitt 1.9) und eine Reihe von Verfahren wurde entwickelt [Muggleton, 1992] [10].

[10] Eine gute Einführung bietet [Nienhuys-Cheng und Wolf, 1997].

Neben der Möglichkeit Relationen von Relationen auszudrücken, zeichnet sich die induktive logische Programmierung dadurch aus, dass sie Hintergrundwissen in das Lernen einbezieht. Die Funktionslernaufgabe der induktiven logischen Programmierung ist:

Gegeben: positive und negative Beispiele E in einer Sprache L_E,
Hintergrundwissen B in einer Sprache L_B, wobei $B \cup H \not\models \perp$

Ziel: eine Hypothese H in einer Sprache L_H, so dass

$B, H, E \not\models \perp$ (Konsistenz),

$B, H \models E^+$ (Vollständigkeit),

$\forall e \in E^- : B, H \not\models e$ (Korrektheit)

Meist besteht sowohl B als auch E aus variablenfreien Fakten. L_H ist eine eingeschränkte Prädikatenlogik.

Auch die induktive logische Programmierung kann als Suche aufgefasst werden (s. Abschnitt 1.2). Wenn man die Generalisierungsbeziehung so formalisieren kann, dass es für zwei Klauseln genau eine Generalisierung (bzw. Spezialisierung) gibt, besteht das Lernen darin, Beispiele so lange zu generalisieren, bis alle positiven Beispiele bezüglich des Hintergrundwissens abgedeckt sind bzw. allgemeine Hypothesen so lange zu spezialisieren, dass keine negativen Beispiele mehr bezüglich des Hintergrundwissens abgedeckt sind. Die Generalisierungsbeziehung ordnet den Hypothesenraum so, dass der induktive Schluss als durch Beispiele gesteuerte Suche in diesem Verband von Hypothesen realisiert werden kann. Die Generalisierungsbeziehung ermöglicht obendrein das sichere Beschneiden des Hypothesenraums:

- Wenn beim Generalisieren bereits die Hypothese $C1$ ein negatives Beispiel abdeckt, so wird auch jede Generalisierung von $C1$ mindestens dieses negative Beispiel abdecken – $C1$ braucht daher nicht weiter generalisiert zu werden.
- Wenn beim Spezialisieren bereits die Hypothese $C1$ ein positives Beispiel nicht abdeckt, so wird auch jede Spezialisierung von $C1$ mindestens dies positive Beispiel nicht abdecken – $C1$ braucht daher nicht weiter spezialisiert zu werden.

Die wichtigste Aufgabe ist es also, die Generalisierungsbeziehung zwischen logischen Formeln (Beispielen, Hypothesen) zu beschreiben. Eine Möglichkeit, die Generalisierung zu definieren, verwendet die Implikation. Eine Klausel $C1$ ist genereller als eine andere, $C2$, wenn $C1 \rightarrow C2$ gilt. Um dann eine Generalisierung zu finden, müssen wir die Klausel finden, die die gegebenen Beispiele impliziert. Dies ist schwierig, weil es darauf hinausläuft, die logische Folgerung zwischen Klauseln als Grundlage zu nehmen, die nicht im allgemeinen Fall entscheidbar ist. Deshalb wird die schwächere Subsumtionsbeziehung bevorzugt. Eine Klausel $C1$ ist genereller als eine andere Klausel $C2$, wenn gilt: $C1$ subsumiert $C2$. Bei Literalen ist das einfach. Ein Literal $L1$ subsumiert ein anderes Literal $L2$, wenn es eine Substitution σ gibt, so dass $L1\sigma = L2$. Damit wird Substitution zur Grundlage der Formalisierung von Induktion.

Wir können uns Generalisierung für Klauseln einmal (semantisch) an den Objekten (Daten, Beispielen) einer logischen Struktur deutlich machen. Eine Klausel $C1$ ist genereller als eine andere Klausel $C2$ (geschrieben: $C1 \geq C2$),wenn sie mehr Objekte abdeckt. So ist

zum Beispiel

$$tier(X) \rightarrow saeugetier(X) \geq tier(rex) \rightarrow saeugetier(rex)$$

und

$$tier(X) \rightarrow saeugetier(X) \geq tier(X) \wedge im\_haus(X) \rightarrow saeugetier(X).$$

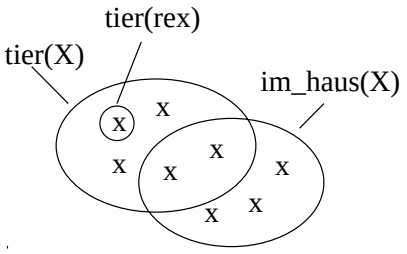

Abbildung 1.10: Objektmengen

Die Säugetiere umfassen einmal nur *rex* (und vielleicht noch andere Objekte), einmal mindestens die Schnittmenge der beiden Objektmengen, schließlich sogar mindestens alle Tiere. An dem Beispiel ist auch deutlich zu sehen, dass neben der Subsumtion auch die logische Folgerung gilt. Zum anderen können wir uns die Generalisierung aber auch (syntaktisch) an der Gestalt der Klauseln klarmachen. Wenn wir sie als Mengen schreiben, so ist die generellere Klausel eine Teilmenge der spezielleren.

$$\{\neg tier(X), \neg im\_haus(X), saeugetier(X)\} \supseteq \{\neg tier(X), saeugetier(X)\}.$$

Natürlich müssen die Terme so substituiert werden, dass es passt.

$$C1 \geq C2 \text{ gdw. } C2 \supseteq C1\sigma.$$

Die Faustregel lautet: je mehr Literale eine Klausel hat, desto spezieller ist sie. Wenn aber durch eine Theorie gegeben ist, dass einige Literale gleichbedeutend sind mit einem anderen Literal, so hilft das einfache Abzählen nichts. Wenn für alle Tiere bekannt ist, dass sie sterblich sind, so wird eine der oben angeführten Klauseln über Tiere nicht spezieller, wenn $sterblich(X)$ hinzugefügt wird. Es ist einfach redundant. Diesen Gedanken werden wir in den nächsten beiden Abschnitten vertiefen.

1.7.0.1 θ-Subsumtion

Wenn wir Lernen in Logik als Suche in einem geordneten Verband von Hypothesen auffassen, brauchen wir eine handhabbare Definition der Generalisierungsbeziehung. Obendrein wollen wir nicht irgendwelche Generalisierungen für zwei Formeln finden, sondern möglichst nur eine. Wenn die Suche nach der Formel, die den zu lernenden Begriff darstellt, schrittweise erfolgen soll, brauchen wir kleine Generalisierungsschritte, um den Zielbegriff nicht

zu überspringen. Gordon Plotkin stellt ein Verfahren vor, wie in der uneingeschränkten Prädikatenlogik erster Stufe die speziellste Generalisierung (*least general generalization* – lgg oder *most specific generalization* – MSG) gefunden werden kann [Plotkin, 1970]. Dabei stellt er zunächst ein Verfahren vor, das für Literale und Terme (zusammengefasst unter der Bezeichnung *Wort*) eine speziellste Generalisierung findet.

$$W1 \geq W2, \text{ wenn } W1\sigma = W2.$$

Zum Beispiel ist $p(X, X, f(g(Y))) \geq p(l(3), l(3), f(g(X)))$, mit $\sigma = \{X/l(3), Y/X\}$. Wieder ist $\geq$ eine Halbordnung (also reflexiv und transitiv).

> Für eine Menge von Wörtern K ist W die speziellste Generalisierung, gdw.:
> Für jedes Wort V aus K gilt: $W \geq V$ und
> wenn für jedes V aus K gilt $W1 \geq V$, dann gilt auch $W1 \geq W$.

Mit der zweiten Bedingung wird gewährleistet, dass es sich bei W um die speziellste Generalisierung handelt. Wenn obendrein $W \geq W1$ gilt, dann sind W1 und W äquivalent. Plotkins Verfahren für die Generalisierung von Wörtern wählt zunächst solche Wörter aus, die dasselbe Prädikatensymbol mit demselben Vorzeichen (negiert oder nicht negiert) haben. Dann muss es eine Substitution für die Terme finden, die an derselben Stelle des Prädikats auftreten, aber verschieden sind. Es wählt eine sonst nicht vorkommende Variable, setzt sie für die Terme ein und schreibt die Substitutionen in die Substitutionslisten der Wörter. Dies wird für alle verschiedenen Terme an selber Argumentstelle so gemacht. Wenn es keine solchen Terme (mehr) gibt, so ist das durch die Substitutionen entstandene Wort die speziellste Generalisierung der Wörter.

Beispiel:

> $V1 : p(f(a(), g(Y)), X, g(Y))$
> $V2 : p(h(a(), g(X)), X, g(X))$
> Y und X nehmen dieselbe Argumentstelle ein und werden durch die neue Variable Z ersetzt. Das ergibt
> $V1 : p(f(a(), g(Z)), X, g(Z))$
> $V2 : p(h(a(), g(Z)), X, g(Z))$
>
> und die Substitutionsliste für $V1$ ist $\{Z/Y\}$, die für $V2$ ist $\{Z/X\}$.
>
> Als nächstes sind $f(a(), g(Z))$ und $h(a(), g(Z))$ verschiedene Terme an derselben Argumentstelle. Sie werden durch eine neue Variable U ersetzt.
> $V1 = V2 = p(U, X, g(Z))$ mit den Substitutionslisten:
> $\sigma 1 = \{U/f(a(), g(Y)), Z/Y\}$ für $V1$ und
> $\sigma 2 = \{U/h(a(), g(X)), Z/X\}$ für $V2$.
> $p(U, X, g(Z))$ ist die speziellste Generalisierung für $V1$ und $V2$.
>
> Durch Anwendung der Substitutionen auf die Generalisierung erhält man wieder $V1$ respektive $V2$ [11].

[11] Statt beim Einfügen einer Ersetzung in die Substitutionsliste die bisherigen Substitutionen auszuführen, kann man auf die Ersetzung in eingebetteten Termen verzichten wie es die Antiunifikation in Prolog tut.

Um dieses Verfahren auf Klauseln übertragen zu können, müssen zunächst alle Literale miteinander kombiniert werden. So macht man aus einer Menge eine äquivalente Liste. Wenn etwa die Klausel $C1$ die Literale $L11$, $L12$, $L13$ enthält und die Klausel $C2$ die Literale $L21$, $L22$, $L23$, so erhält man die Listen

$C1 : [L11, L11, L11, L12, L12, L12, L13, L13, L13]$
$C2 : [L21, L22, L23, L21, L22, L23, L21, L22, L23]$

Diese Listen können nun einfach elementweise gegeneinander abgeglichen werden. Dabei kann man alle übereinander stehenden Literalpaare streichen, die nicht dasselbe Prädikatssymbol haben. Für die resultierenden Listen findet Plotkin Generalisierungen mit Hilfe der θ-Subsumtion.

Plotkins Übertragung des Verfahrens für Wörter auf Klauseln ist einfach: er formt Klauseln so um, dass neue Funktionssymbole anstelle der Prädikatssymbole stehen und verfährt mit den Funktionen wie oben beschrieben. Man behandelt die Menge von Literalen einer Klausel einfach als Terme eines Prädikats!

> Die Menge der Literale in den Klauseln werden mit zwei Indizes versehen, einer für die Klausel (i, j), einer für die Literale selbst $(1, .., l, .., n)$. Jedes Literal L_{jl} hat ja die Form $(\pm)p_l(t_1, \ldots, t_k)$.
> Bei n Literalen einer Klausel wird ein neues, n-stelliges Prädikat q konstruiert, das für jedes k-stellige Prädikat der Klausel eine neue k-stellige Funktion als Argument bekommt:
> $q(f_1(t_{j11}, .., t_{jk1}), \ldots, f_n(t_{j1n}, \ldots, t_{jkn}))$
> Damit erhält man für zwei Klauseln zwei solche q-Literale, deren Terme generalisiert werden. Das Ergebnis hat dann die Form:
> $q(f_1(u_{11}, .., u_{k1}), \ldots, f_n(u_{1n}, \ldots, u_{kn}))$
> Dann ist $\{(\pm)p_l(u_{11}, .., u_{k1}), \ldots, (\pm)p_n(u_{1n}, \ldots, u_{kn})\}$ die speziellste Generalisierung der zwei Klauseln.

Wir haben gesehen, dass die Listen groß werden. Die Reduktion entfernt überflüssige Literale aus Klauseln. Alle solche L aus einer Klausel E werden entfernt, für die es eine Substitution gibt, so dass $E \setminus \{L\} \supseteq E\sigma$. Auch ohne L umfasst E immer noch $E\sigma$. Da aber alle möglichen Teilmengen der Klausel gebildet werden müssen, ist die Reduktion exponentiell.

Diese Formalisierung hat zwei Schwächen: sie ist ineffizient und sie berücksichtigt kein Hintergrundwissen. Deshalb hat Wray Buntine eine generalisierte θ-Subsumtion eingeführt, mit der die Induktion einer Klausel aus einer anderen Klausel bezüglich einer Theorie beschrieben werden kann [Buntine, 1988].

1.7.0.2 Generalisierte θ-Subsumtion

Die generalisierte θ-Subsumtion generalisiert zwei funktionsfreie Hornklauseln bezüglich gegebenem Hintergrundwissen. Bei Hornklauseln nennen wir das positive Literal den Klauselkopf, die negativen Literale den Klauselkörper. Wir beziehen uns auf den Klauselkopf der Klausel $C1$ mit $C1_{kopf}$, entsprechend schreiben wir $C1_{körper}$ für den Klauselkörper von $C1$. Die Substitution θ ist nun derart, dass für alle Variablen des Klauselkopfes neue Kon-

stanten eingeführt werden. Wir belassen σ als Bezeichner für (irgend)eine Substitution von Termen. Dann definiert Buntine die Generalisierung folgendermaßen [Buntine, 1988]:

$C1 \geq_B C2$ gdw.
$\exists\sigma$, so dass $C1_{kopf}\sigma = C2_{kopf}$ und
$B, C2_{körper}\theta \models \exists(C1_{körper}\sigma\theta)$

Wir müssen also die generellere Klausel durch Substitutionen erst einschränken, damit sie aus der spezielleren folgt. Diese Definition können wir für alle Klauseln einer Menge von Klauseln anwenden, d.h. um Mengen von Klauseln zu vergleichen, beschränken wir uns auf den Vergleich aller einzelnen Klauseln der Mengen.

Die Definition operationalisiert Buntine, indem er für jede Klausel C_i der generelleren Klauselmenge zeigt, dass sie zurückgeführt werden kann auf eine Klausel der spezielleren Klauselmenge, indem

- Variable aus C_i in Konstante oder andere Terme überführt werden,
- Atome dem Klauselkörper von C_i hinzugefügt werden, oder
- der Klauselkörper von C_i im Hinblick auf das Hintergrundwissen B teilweise ausgewertet wird, d.h. ein Atom aus C_i wird mit einer Klausel aus B resolviert.

Dieses Verfahren ist entscheidbar, wenn das Hintergrundwissen keine Funktionen enthält.
Wie sieht es bei den folgenden Klauseln aus, ist $C1 \geq C2$ bezüglich B?

B sei: $tier(X) \wedge im\_haus(X) \rightarrow haustier(X)$
$C1$ sei: $flauschig(Y) \wedge haustier(Y) \rightarrow kuscheltier(Y)$
$C2$ sei: $flauschig(Z) \wedge tier(Z) \wedge im\_haus(Z) \rightarrow kuscheltier(Z)$

Wir erhalten als B, $C2_{körper}\theta$ mit $\theta : \{Z/b\}$:
$tier(X) \wedge im\_haus(X) \rightarrow haustier(X)$
$flauschig(b) \wedge tier(b) \wedge im\_haus(b)$

und folgern mit $\sigma 1 : \{X/b\}$ und der Schnittregel
$flauschig(b) \wedge haustier(b)$ Das ist $C1_{körper}\sigma\theta$, wenn wir σ:{} wählen, $\theta : \{Y/b\}$.

Es gilt also $C1 \geq C2$ bezüglich B.

Das Verfahren von [Buntine, 1988] zählt also nicht nur die Literale durch, sondern berücksichtigt, dass einige Literale (hier: $tier(X)$ und $im\_haus(X)$) durch das Hintergrundwissen verbunden sind mit einem anderen Literal (hier: $haustier(X)$). Die Faustregel ist jetzt mit den Substitutionen σ und θ und der Einbeziehung des Hintergrundwissens so formalisiert, dass sie Ergebnisse liefert, die unserer Intuition entsprechen. Ihre Operationalisierung verwendet die Resolution, um alle Literale herauszuschneiden, die sowieso durch B gegeben sind. Die Substitutionen sorgen dafür, dass keine unzusammenhängenden Objekte einbezogen werden. Dann können wir einfach abzählen, welche Klausel mehr Literale enthält.

1.7.1 Beschränkung der Suche

Die Suche im Hypothesenraum aller logischer Formeln muß beschränkt werden, damit sie effizient wird. Zum einen kann man die Logik und damit den Suchraum einschränken (s. Abschnitt 1.9). Meist gilt die Beschränkung für ein Verfahren in jeder Anwendung. Es gibt

aber auch die Möglichkeit, den Benutzer spezifizieren zu lassen, an welchen Hypothesen er interessiert ist. Es werden dann nur diese Hypothesen vom System untersucht. Dabei soll der Benutzer nicht jede einzelne Hypothese angeben müssen, sondern allgemeiner deklarieren können, welche Mengen von Hypothesen untersucht werden sollen. Das Lernverfahren nimmt die anwendungsspezifische Beschränkung als Eingabe und sucht nur in dem eingeschränkten Hypothesenraum. Die deklarative und veränderbare Beschränkung der Sprache L_H heißt *declarative bias*. Ein Lernverfahren, das eine explizite, syntaktische Sprachbeschränkung durch den Benutzer zulässt, ist RDT [Kietz und Wrobel, 1992]. Der Benutzer gibt hier die syntaktische Form der Regeln an, die vom Lernverfahren betrachtet werden sollen. So kann er angeben, dass er nur an Regeln mit maximal drei Prämissen interessiert ist, etwa weil längere Regeln zu unübersichtlich sind.

Zum anderen kann man den (unbeschränkten) Suchraum unvollständig, heuristisch durchsuchen. Dafür ist das System FOIL ein Beispiel [Quinlan, 1990]. Die Hypothesensprache L_H umfasst funktionsfreie Klauseln, deren Kopf aus genau einem positiven Literal besteht und deren Körper aus positiven oder negierten Literalen besteht. Terme müssen mit dem Klauselkopf verbunden sein. Beispiele werden in Form von variablen- und funktionsfreien Fakten gegeben. Wenn keine negativen Beispiele vorliegen, berechnet FOIL alle möglichen, nicht als positive Beispiele gegebenen Fakten als negative Beispiele (*closed-world assumption*). Das Hintergrundwissen liegt in Form von extensionalen Definitionen von Prädikaten in Form von variablen- und funktionsfreien Fakten vor. Wie RDT erzeugt auch FOIL die Hypothesen von der allgemeinsten zu den spezielleren. Im Gegensatz zu RDT durchsucht FOIL den Hypothesenraum nur dort, wo ein Informationsgewinn erwartet wird. Dazu wird eine Heuristik verwendet, die – ähnlich wie bei ID3 – auf der Entropie basiert.

FOIL erstellt aus den gegebenen Beispielen für ein k-stelliges Zielprädikat eine Menge von k-stelligen Tupeln, wobei jedes Tupel eine Wertebelegung der k Variablen angibt. Die Tupel sind als positive oder negative Beispiele klassifiziert. Die Tupelmenge ist vollständig, d.h. alle möglichen Wertebelegungen kommen entweder als positives oder als negatives Beispiel vor. Der Algorithmus besteht aus zwei Schleifen, einer äußeren und einer inneren. Die äußere Schleife findet für positive Beispiele bzw. Tupel T^+ eine Klausel, die sie abdeckt. Die abgedeckten positiven Beispiele werden aus der Tupelmenge entfernt und für die verbleibenden positiven Beispiele wird eine weitere Klausel gelernt. Es werden so viele Klauseln gelernt, dass insgesamt von der Menge der Klauseln alle positiven Beispiele abgedeckt werden.

Bis T^+ keine Tupel mehr enthält:

- finde eine Klausel, die positive Beispiele abdeckt,
- entferne alle Tupel aus T^+, die von dem Klauselkörper abgedeckt werden.

In der inneren Schleife wird der Körper einer Klausel durch Literale erweitert, die Klausel also spezialisiert. Ein zusätzliches Literal wird aufgrund des Informationsgewinns ausgewählt, den es für die Charakterisierung des Zielprädikats erbringt.

- Initialisiere T_i mit den Beispielen in Tupeldarstellung und setze $i = 1$
- Solange T^- Tupel enthält:
  - finde ein nützliches Literal L_i und füge es dem Klauselkörper hinzu,

- erstelle eine neue Tupelmenge T_{i+1}, die nur diejenigen Beispiele enthält, für die L_i gilt. Falls L_i eine neue Variable einführt, werden die Tupel um diese erweitert. Die Klassifikation der Tupel in T_{i+1} bleibt diejenige von T_i.
- setze i auf $i + 1$

Die Information, die man bei einer Tupelmenge T_i mit p_i positiven und n_i negativen Beispielen braucht, um zu entscheiden, ob ein Tupel positiv ist, ist

$$I(T_i) = -log_2 \frac{p_i}{p_i + n_i}$$

Ziehen wir den Informationsgehalt der neuen Tupelmenge von dem der alten Tupelmenge ab, so erhalten wir den Informationsgewinn durch das neu eingeführte Literal. Dieser Informationsgewinn wird durch die Anzahl der positiven Tupel, die durch das neue Literal abgedeckt werden, p_{i+1}, gewichtet. Der gewichtete Informationsgewinn durch ein neues Literal L_i bezogen auf die Tupelmenge T_i wird also durch die folgende Formel ausgedrückt:

$$Gain(L_i) = (p_{i+1} + n_{i+1})(I(T_i) - I(T_{i+1}))$$

Ein einfaches Beispiel ist das Lernen der Relation *tochter* aus den folgenden Beispielen, wobei vor dem Semikolon die positiven Tupel, nach dem Semikolon die negativen Tupel angegeben werden. Das Hintergrundwissen ist durch die Relationen *eltern* und *weibl* gegeben. Das Beispiel ist [Lavrač und Džeroski, 1994] entnommen.

tochter(name,name)	eltern(name,name)	weibl(name)
mary, ann	ann, mary	mary
eve, tom ;	ann, tom	ann
tom, ann	tom, eve	eve
eve, ann	tom, ian	

Die erste Menge T_1 enthält alle 4 Tupel, $p_1 = 2, n_1 = 2$. Ihr Informationsgehalt ist $-log_2 \frac{2}{4} = 1$. Die Klausel enthält nur das Zielprädikat $tochter(X, Y)$ als Klauselkopf. Wird nun $weibl$ als erstes Literal in den Klauselkörper eingetragen, so enthält die neue Tupelmenge T_2 nicht mehr das negative Beispiel tom, ann, da $weibl(tom)$ nicht gilt. Damit ist die neue Tupelmenge geordneter: $I(T_2) = 0,58$. Alle positiven Beispiele bleiben abgedeckt. Der Informationsgewinn von $weibl$ ist also:

$$Gain(weibl(X)) = 2(1 - 0,58) = 0,84$$

Wird nun noch $eltern(Y, X)$ als nächstes Literal hinzugefügt, so brauchen wir bei der resultierenden Tupelmenge $T3$ keine weitere Information für die Klassifikation: $I(T_3) = 0$. Der Informationsgewinn von $eltern$ ist:

$$Gain(eltern(Y, X)) = 2(0,58 - 0) = 1,16$$

Die Klausel $eltern(Y, X) \wedge weibl(X) \rightarrow tochter(X, Y)$ deckt alle positiven Beispiele ab und kein negatives. Da in der Menge T_3 kein negatives Beispiel mehr enthalten ist, endet die innere Schleife. Damit ist der erste Schritt der äußeren Schleife abgeschlossen und es

werden alle positiven Beispiele aus der Menge T_1 entfernt, die von der Klausel abgedeckt werden. Da dies alle positiven Beispiele sind, endet auch die äußere Schleife.

Da FOIL den Hypothesenraum gemäss der Heuristik des Informationsgewinns durchsucht, kann es eine richtige Lösung des Lernproblems verpassen. Dies ist der Fall, wenn ein Literal mit geringem Informationsgewinn gewählt werden muss, das eine neue Variable einführt, die für die Wahl geeigneter weiterer Literale nötig ist. In sehr vielen Fällen führt die Heuristik jedoch dazu, dass FOIL sehr schnell die richtige Lösung findet.

1.8 Adaptivität und Revision

Die Generalisierungsbeziehung zwischen Klauseln strukturiert den Hypothesenraum. Wir haben nun Generalisierungsoperatoren kennengelernt, die von zwei Beispielen oder zwei spezielleren Hypothesen eine minimal generellere Hypothese erzeugen. Diese Generalisierungsoperatoren durchsuchen den Hypothesenraum *bottom-up*. Die Verfahren FOIL und RDT durchsuchen den Hypothesenraum *top-down*. Beide verwenden also Spezialisierungsoperationen. Das System MIS bietet zwei Spezialisierungsoperationen an: eine bedarfsgestützte, die bei Widersprüchen eine falsche Aussage aus der Theorie löscht (*contradiction backtracing*), und eine Verfeinerungsoperation, die eine falsche und deshalb verworfene Aussage spezialisiert, bis die Spezialisierung korrekt ist [Shapiro, 1981], [Shapiro, 1983]. In diesem Abschnitt sollen Spezialisierungsoperatoren ausführlich behandelt werden. In Abschnitt 1.8.1 wird das Problem der Vollständigkeit eines Verfeinerungsoperators behandelt. Kann man wirklich von der leeren Klausel ausgehend alle Klauseln durch eine Folge von Operator-Anwendungen erreichen? Eine bedarfsgestützte minimale Spezialisierung einer Theorie wird in Abschnitt 1.8.2 beschrieben. Interessant ist dabei, daß hier die Spezialisierungsoperation mit dem Bereich des nicht-monotonen Schließens in Beziehung gesetzt wird.

1.8.1 Vollständige Spezialisierung unter θ-Subsumtion

Um effizienter zu verfeinern, hat Ehud Shapiro in seinem System MIS den Verfeinerungsoperator so definiert, daß er mit reduzierten Formeln arbeitet [Shapiro, 1981]. Unter θ-Subsumtion sind Formeln reduziert, wenn sie keine redundanten Literale enthalten (siehe Abschnitt 1.7.0.1). Als weitere Einschränkung war die Verfeinerung *sr* in MIS stets *umfangreicher* als das Verfeinerte. Insbesondere durfte die Anzahl der Literale bei der Verfeinerung nicht sinken. Was zunächst einleuchtend aussieht, führt leider zu einem Problem, das Patrick van der Laag und Shan-Hwei Nienhuys-Cheng entdeckten [van der Laag, 1995][Nienhuys-Cheng *et al.*, 1993]. Wenn wir bei der Verfeinerung nur reduzierte Formeln zulassen und keine kleiner wird als ihr Vorgänger, gibt es zwischen einigen reduzierten Formeln keine Verfeinerungsbeziehung, obwohl die eine Formel die andere subsumiert. Um wirklich alle Formeln erreichen zu können, müssen entweder auch nicht-reduzierte Formeln in dem Verfeinerungsgraphen als Knoten vorkommen dürfen. Oder der Verfeinerungsoperator darf eine Formel verkürzen. Zwei Beispiele zeigen das Problem.

Seien P und Q zwei reduzierte Formeln, wobei P allgemeiner ist als Q.
$P : \{p(a, W), p(X, b), p(c, Y), p(Z, d)\}$
$Q : \{p(a, b), p(c, b), p(c, d), p(a, d)\}$

Es gibt einen Pfad zwischen P und Q, aber er enthält nicht reduzierte Formeln.
$P_1 : \{p(a, b), p(X, b), p(c, Y), p(Z, d)\}$
$P_2 : \{p(a, b), p(c, b), p(c, Y), p(Z, d)\}$
$P_3 : \{p(a, b), p(c, b), p(c, d), p(Z, d)\}$
$Q : \{p(a, b), p(c, b), p(c, d), p(a, d)\}$

Alle Substitutionen, die notwendig sind, um mit *sr* schrittweise von P nach Q zu kommen, führen über nicht-reduzierte Klauseln. Eine nicht-reduzierte Klausel θ-subsumiert einen Teil von sich selbst. So ist z.B. die reduzierte Fassung von P_1

$P_1' : \{p(a, b), p(c, Y), p(Z, d)\}$

Das Literal $p(X, b)$ ist redundant unter der θ-Subsumtion (siehe Abschnitt 1.7), weil $P_1\theta \subseteq P_1 \setminus \{p(X, b)\}$ mit $\theta = \{X/a\}$. Diese kürzere Fassung darf laut Shapiro nicht eingeführt werden. Tatsächlich ist am Ende Q nicht kürzer als P. Ohnehin ist eine Spezialisierung manchmal kürzer:

$\{p(X, Y), p(Y, X)\}$ subsumiert $\{p(X, X)\}$

In diesem speziellen Falle hatte Shapiro die Gleichheit als zusätzliches Literal angehängt:

$\{p(X, Y), p(Y, X), Y = X\}$

Das Problem hängt nicht nur mit der Länge von Klauseln zusammen, wie das nächste Beispiel zeigt, bei dem auf dem Weg zwischen einer kurzen, allgemeineren Klausel und ihrer längeren Spezialisierung eine nicht-reduzierte Klausel liegt.

$P : \{p(X), \neg q(X, a)\}$
$Q : \{p(X), \neg q(X, a), \neg q(Y, Z), \neg q(Z, Y)\}$

Nun ist einer der Zwischenschritte bei der Verfeinerung mit *sr* eine nicht-reduzierte Klausel, nämlich R.
$R : \{p(X), \neg q(X, a), \neg q(Y, Z)\}$

Mit $\theta = \{Y/X, Z/a\}$ fallen die beiden letzten Literale zusammen und $R\theta \subseteq R \setminus q(Y, Z)$, d.h.: R ist nicht-reduziert.

Das Beispiel ist gerade so konstruiert, dass die *beiden* letzten Literale in Q gemeinsam unmöglich machen, eines davon zu streichen.

Die Beispiele zeigen, dass der Verfeinerungsoperator von Ehud Shapiro unvollständig ist, weil er den Klauselgraph auf reduzierte und immer umfangreicher werdende Klauseln einschränkt. Andererseits sind seine Motive für die Einschränkung ja überaus einleuchtend – irgendwie muss die Verfeinerung eingeschränkt werden, um handhabbar zu sein. Wir müssen also aus den Gegenbeispielen zu *sr* lernen, wie die Verfeinerung zu verbessern ist. Dazu verwenden van der Laag und Nienhuys-Cheng zwei Hilfsmittel: $eq(P)$ und die Idee des *generellsten Literals*.

Definition 1.8.1 (eq(P)) *Die Funktion $eq(P)$ liefere alle zu P äquivalenten und höchstens um k Literale erweiterten Klauseln.*

Das *generellste Literal* hilft bei der Auswahl, welches Literal zu der Klausel bei der Spezialisierung hinzugefügt werden soll.

Definition 1.8.2 (Generellstes Literal bezgl. einer reduzierten Klausel) *Für eine reduzierte Klausel C heißt ein Literal L das generellste Literal bezüglich C, wenn $C \cup \{L\}$ reduziert ist und für jedes andere Literal M, das echt genereller ist als L bezüglich C, ist $C \cup \{M\}$ nicht reduziert.*

Es geht bei dieser Definition um die sinnvolle Erweiterung einer Klausel bei der Spezialisierung. Die erweiterte Klausel soll im Raum der reduzierten Klauseln gerade "einen Schritt" unter der Ausgangsklausel liegen.

Ein Beispiel:

> Sei $C : q(X) \leftarrow p(X,Y)$. Wenn wir jetzt erweitern zu $C' : q(X) \leftarrow p(X,Y), p(U,X)$, so gibt es kein θ das die beiden Literale im Klauselkörper zusammenfallen lässt. Allerdings ist die Generalisierung von $p(U,X)$ bezüglich C zu $p(U,V)$ nicht mehr durch eine gemeinsame Variable mit dem Rest der Klausel beschränkt und $C'' : q(X) \leftarrow p(X,Y), p(U,V)$ kann mit $\theta = \{U/X, V/Y\}$ reduziert werden. Das Literal $L : p(U,X)$ ist also ein generellstes Literal bezüglich C, während das Literal $M : p(U,V)$ zu allgemein ist.

Nun haben wir oben gesehen, dass die Beschränkung auf den Raum der reduzierten Klauseln die Spezialisierung unvollständig macht. Deshalb beziehen van der Laag und Nienhuys-Cheng die äquivalenten, nicht-reduzierten Klauseln der Länge m mit ein. Diese äquivalenten Klauseln werden durch die *inverse Reduktion* gefunden. Es ist einigermaßen einfach, eine Reduktion rückgängig zu machen: an die reduzierte Klausel C muss mindestens ein Literal L angehängt werden mit einem Prädikatensymbol und Vorzeichen, das in C vorkommt. Die Argumente in L müssen teilweise disjunkt von den in C vorkommenden Variablen sein, damit sie bei der Reduktion via θ mit einer in C vorkommenden Variable ersetzt werden können. Dann können wir Varianten außer acht lassen und kommen zu allen möglichen θ-Substitutionen. Das einzige Problem bei der inversen Reduktion ist, dass wir beliebig viele Literale für jedes Literal in C anhängen könnten. Deshalb wird für die äquivalenten nicht-reduzierten Klauseln eine Längenbeschränkung eingeführt. Es dürfen nur k zusätzliche Literale angehängt werden.

Ein Beispiel:

> $C : \{p(X,X)\}$, $k = 1$
> $eq(C) : \{\{p(X,X), p(X,Y)\}, \{p(X,X), p(Y,X)\}, \{p(X,X), p(Y,Z)\}, \{p(X,X), p(Y,Y)\}\}$
>
> Hier wurden Y und Z als die nicht in C vorkommenden Variablen gewählt – jeder andere Bezeichner stellt nur eine Variante dar. Mit $\theta = \{Y/X\}$ bzw. $\theta = \{Y/X, Z/X\}$ wird jede Klausel in $eq(C)$ wieder reduziert.

Der neue Verfeinerungsoperator sr_τ unterscheidet sich jetzt von dem unvollständigen prinzipiell nur darin, dass auch auf die Klauseln in $eq(C)$ die Substitutionen des Verfeinerungsoperators angewandt werden. Der besseren Lesbarkeit halber ist die Substitution der Variablen in einem Atom aufgeteilt in die Substitution durch andere Variablen und durch Funktionen (einschließlich Konstanten). Die Bedingung, dass P die Spezialisierung Q $\theta-$subsumiert war in sr implizit.

Definition 1.8.3 (Schrittweise Verfeinerungsrelation $sr_\tau(P,Q)$) *Für zwei Klauseln in der Sprache L gilt die Verfeinerungsrelation $sr_\tau(P,Q)$ gdw.*

1. *P $\theta-$subsumiert echt Q und es gibt Klauseln $P' \in eq(P)$ und $Q' \in eq(Q)$, so dass $Q' = P'\theta$, wobei $\theta = \{X/Y\}$, und sowohl X als auch Y kommen in P' vor; oder*
2. *P $\theta-$subsumiert echt Q und es gibt Klauseln $P' \in eq(P)$ und $Q' \in eq(Q)$, so dass $Q' = P'\theta$, wobei $\theta = \{X/f(Y_1,\ldots,Y_n\}$ und X kommt in P' vor und die Y_i sind verschiedene, nicht in P' vorkommende Variablen; oder*
3. *$Q = P \cup \{M\}$, wobei M das generellste Literal bezüglich P ist, das bezüglich Prädikatensymbol oder Vorzeichen von jedem Literal in P verschieden ist.*

Ein Beweis für die Vollständigkeit von sr_τ findet sich in [van der Laag, 1995]. Der dritte Fall erlaubt nur endlich viele generellste Literale, die angehängt werden können. Der erste und zweite Fall wendet Substitutionen auch auf die nicht-reduzierten Klauseln an, um die Verfeinerungsrelation zwischen ihren reduzierten Äquivalenten herzustellen. Wir können uns nun an den obigen Gegenbeispielen gegen sr klar machen, dass sr_τ nicht an ihnen scheitert.

$P : \{p(a,W), p(X,b), p(c,Y), p(Z,d)\}$
$Q : \{p(a,b), p(c,b), p(c,d), p(a,d)\}$

Im ersten Schritt wenden wir den zweiten Fall an, wobei
$P'_1 : \{p(a,b), p(X,b), p(c,Y), p(Z,d)\}, \theta = \{W/b\}$ und
$P_1 : \{p(a,b), p(c,Y), p(Z,d)\}$

Tatsächlich θ-subsumiert P echt P_1, d.h. mit $\theta = \{W/b, X/a\}$ gilt $P\theta \subseteq P_1$, wobei P und P_1 nicht äquivalent sind. Setzen wir $P' = P$ und nehmen $P'_1 \in eq(P_1)$, dann sehen wir, dass $P'_1 = P'\theta$ mit $\theta = \{W/b\}$, wobei W in P' vorkommt und die nullstellige Funktion b natürlich keine Variable aus P' enthält (siehe Fall 2). Wir stellen außerdem fest, dass P_1 auf dem richtigen Pfad liegt, denn es θ-subsumiert Q, d.h. z.B. mit $\theta = \{Y/b, Z/a\}$ gilt $P_1\theta \subseteq Q$.

Im nächsten Schritt wenden wir den zweiten Fall auf P'_1 an und erhalten mit $\theta = \{X/c\}$:
$P'_2 : \{p(a,b), p(c,b), p(c,Y), p(Z,d)\}$, wobei $P'_2 \in eq(P_2)$.
$P_2 : \{p(a,b), p(c,Y), p(Z,d)\}$

P_1 subsumiert P_2 und $P'_2 = P'_1\theta$, wobei X in P'_1 vorkommt und c nicht.

Es ergibt sich dieselbe Kette wie oben und alle reduzierten Klauseln liegen auf dem Pfad von P nach Q. Sie werden aber gefunden über ihre nicht-reduzierten Äquivalente.

Auch das zweite Gegenbeispiel gegen sr wird von sr_τ korrekt in eine Verfeinerungsrelation gestellt.

$P : \{p(X), \neg q(X,a)\}$
$Q : \{p(X), \neg q(X,a), \neg q(Y,Z), \neg q(Z,Y)\}$

Wir beginnen mit $P' \in eq(P)$:
$P' : \{p(X), \neg q(X,a), \neg q(Y,Z), \neg q(U,V)\}$

P' ist θ-äquivalent zu P. Nun wenden wir auf P' den ersten Fall von sr_τ an und erhalten mit der Substitution von U durch Z:
$P_1 : \{p(X), \neg q(X,a), \neg q(Y,Z), \neg q(Z,V)$

Hierauf wenden wir noch einmal den ersten Fall an mit $\theta = \{V/Y\}$ und sind dann richtig bei Q angekommen.

Obwohl die Pfade im Verfeinerungsgraphen stets endlich sind, ist der Graph *breiter* geworden: es müssen viel mehr Alternativen (nämlich alle äquivalenten Klauseln) untersucht werden. Setzen wir k zu klein, so ist die Vollständigkeit nicht gewährleistet (z.B. bei $k = 1$ im vorangegangenen Beispiel). Insofern ist sr_τ weniger von praktischem Wert als vielmehr erhellend für die Schwierigkeiten der Spezialisierung in Prädikatenlogik.

1.8.2 Minimale Spezialisierung

Wir haben eben die Spezialisierung *einer* Klausel betrachtet. Wie aber sieht es aus, wenn wir eine Menge von Klauseln spezialisieren wollen? Dazu muss man erst einmal diejenigen Klauseln in der Theorie finden, die Anlass zu einer Spezialisierung geben. Es gilt also die Klauseln zu finden, die daran schuld sind, dass ein negatives Beispiel abgedeckt wird. Anders ausgedrückt: es gilt, die Klauseln zu finden, die für einen Widerspruch verantwortlich sind. Diese Umformulierung zeigt, warum der Spezialisierungsschritt, der im maschinellen Lernen vorgenommen wird, um negative Beispiele auszuschliessen, so bedeutsam für viele Anwendungen ist. Es ist derselbe Schritt, der vorgenommen werden muss, wenn eine Wissensbasis inkonsistent geworden ist. Und das werden Wissensbasen im Laufe ihrer Verwendung immer: sei es, dass sich die Welt verändert, sei es, dass sich das Wissen über die Welt verändert! Es ist also gar nicht verwunderlich, dass es eine Fülle von Untersuchungen zum Thema der Wissensveränderung (Stichwort: *theory of change*) gibt – *knowledge in flux*, wie Peter Gärdenfors sein wichtiges Buch nannte. Damit der Zusammenhang zwischen maschinellem Lernen und dem Bereich des nicht-monotonen Schließens deutlich wird, fassen wir die Schwierigkeiten der Wissensveränderung und die Beiträge dazu aus dem maschinellen Lernen hier zusammen.

Definition 1.8.4 (Basis und Hülle einer Theorie) *Eine Theorie, in Prädikatenlogik dargestellt, besteht aus dem, was tatsächlich aufgeschrieben ist, der* Basis B*, und dem, was aus der Basis folgt, der inferentiellen Hülle* $Cn(B)$*.*

Definition 1.8.5 (Abgeschlossenheit einer Theorie) *Eine Theorie ist abgeschlossen, wenn* $Cn(B) = B$*.*

Ein Beispiel:

$B : \{p(a),$
$\quad q(X) \leftarrow p(X)\}$

Wenn die Signatur nur a, b, p, q bereithält, dann ist
$Cn(B) : \{p(a), q(X) \leftarrow p(X), p(a) \vee p(b), p(a) \vee \neg p(b), \ldots\}$.

Das Problem der **Revision** besteht darin, unerwünschte Fakten (die hier ruhig allquantifizierte Variablen enthalten dürfen) auszuschließen:

Definition 1.8.6 (Revision) *Sei das folgende gegeben:*

- *eine Theorie B*
- *eine Menge von Fakten $F = \{f_1, \ldots f_n\}$*

Ziel ist es, eine revidierte Theorie $B \frown F$ zu finden, so dass $Cn(B \frown F) \cap F = \emptyset$.

Mit der einleuchtenden Annahme, dass eine gute Revision so viel unangetastet lässt, wie es beim Ausschluss der unerwünschten Fakten möglich ist, kommt man zu der *maximal generellen und korrekten Spezialisierung* als Ziel der Revision [Muggleton und Bain, 1992].

Definition 1.8.7 (Maximal generelle korrekte Spezialisierung) *Eine Theorie B' ist eine maximal generelle korrekte Spezialisierung von B bezüglich F, wenn*

- $Cn(B) \supseteq Cn(B')$,
- $F \notin Cn(B')$,
- *für alle anderen B'' gilt: wenn $Cn(B) \supseteq Cn(B'') \supseteq Cn(B')$, dann $F \in Cn(B'')$.*

Wenn eine Theorie also genereller ist als die maximal generelle, dann leitet sie F ab. Nun gibt es im Bereich des nicht-monotonen Schließens eine Reihe von Erkenntnissen über das Verhalten von abgeschlossenen Theorien. Die Postulate von Peter Gärdenfors [Gärdenfors, 1988] beschreiben eine Klasse von Revisionsoperatoren, über die man einiges weiß.

Gärdenfors Postulate: Die Spezialisierung bei abgeschlossenen Theorien soll den folgenden Postulaten gehorchen.

1. $B \frown f$ sei eine abgeschlossene Theorie.
2. $B \frown f \subseteq B$ (Inklusion)
3. Wenn $f \notin B$, dann $B \frown f = B$
4. Wenn $f \notin Cn(\emptyset)$, dann $f \notin B \frown f$ (Erfolg)
5. Wenn $Cn(f) = Cn(g)$, dann $B \frown f = B \frown g$ (Erhalt)
6. $B \subseteq Cn(B \frown f \cup \{f\})$ (Wiederherstellung)

Nun hat Bernhard Nebel festgestellt, dass alle Revisionsoperatoren, die den Gärdenfors-Postulaten gehorchen, mithilfe der maximalen korrekten Teilmengen einer abgeschlossenen Theorie definiert werden können [Nebel, 1989]. Diese Teilmengen sind die möglichen Spezialisierungen, wenn f ausgeschlossen werden soll. Es ist so eine Art Versionenraum für die Revision. Man spricht von diesen alternativen Möglichkeiten als B *ohne* f, geschrieben $B \downarrow f$.

Definition 1.8.8 (Mögliche Spezialisierungen $B \downarrow f$, um f auszuschließen) *Sei B eine Theorie, f ein auszuschließendes Fakt, dann ist*

$$B \downarrow f := \{B' \subseteq B \mid f \notin Cn(B') \text{ und für alle } B'' \text{ gilt: wenn } B' \subset B'' \subseteq B, \text{ dann } f \in Cn(B'')\}$$

Für das Beispiel oben ist, wenn wir $f = q(a)$ entfernen wollen, $B \downarrow f$:

$$B \downarrow q(a) = \left\{ \begin{array}{l} \{p(a), p(a) \vee p(b), p(a) \vee \neg p(b), \ldots \} \\ \{q(X) \leftarrow p(X), p(a) \vee p(b), \ldots \} \\ \{q(X) \leftarrow p(X), p(a) \vee \neg p(b), \ldots \} \\ \ldots \end{array} \right\}$$

Nun können wir alle den Gärdenfors-Postulaten gehorchenden Revisionsoperatoren einheitlich darstellen, wobei sie jeweils unterschiedliche Auswahlen aus den maximalen Teilmengen treffen mögen.

Satz 1.8.1 (Revisionsoperation [Alchourron *et al.*, 1985]) *$\frown$ ist eine Revisionsoperation, die die Gärdenfors-Postulate befolgt, gdw. es eine Selektionsfunktion γ gibt, die über den maximalen Teilmengen $B \downarrow f$ arbeitet, d.h. $\gamma(B \downarrow f) \subseteq B \downarrow f$, so dass*

$$B \frown f = \left\{ \begin{array}{ll} \bigcap \gamma(B \downarrow f) & \textit{wenn } f \notin Cn(\emptyset) \\ B & \textit{ansonsten.} \end{array} \right\}$$

Das Ergebnis der Revision ist also die Schnittmenge der von γ ausgewählten maximalen Teilmengen.

Definition 1.8.9 (Maxi-choice contraction) *Wenn γ nur eine Menge auswählt, so handelt es sich um eine maxi-choice contraction.*

Diese Revisionsoperation entspricht ganz genau der Spezialisierungsoperation von Stephen Muggleton,[12] die eine maximal generelle korrekte Spezialisierung liefert. Das ist deshalb schade, weil gerade über die *maxi-choice contraction* Unangenehmes bekannt ist:

- Sie liefert eine abgeschlossene Theorie zurück. Damit ist sie für praktische Anwendungen nicht mehr relevant, denn nur bei Spielbeispielen im Skript kann man für eine abgeschlossene Theorie eine geeignete Basis finden (d.h. eine Axiomatisierung).
- Wenn wir mit abgeschlossenen Theorien arbeiten, wissen wir nicht mehr, was mal als Basis explizit angegeben war und was abgeleitet wurde. In unserem Beispiel könnten wir also, wenn wir $p(a)$ ungültig machen sollen, das abgeleitete $q(a)$ behalten. Wollen wir das?
- Als *Besserwisser-Effekt* wird bezeichnet, dass bei einer abgeschlossenen Theorie für jedes Fakt entweder das Fakt oder seine Negation gilt: für jede Aussage g gilt entweder $g \in Cn(B \frown f \cup \neg f)$ oder $\neg g \in Cn(B \frown f \cup \neg f)$. Wir wollten eigentlich nur f negieren und haben wundersamerweise neues Wissen über g in unserem Revisionsergebnis.

Wenn wir die Spezialisierung in einem Lernverfahren anwenden wollen, das negative Beispiele ausschließt, oder wenn wir sie einsetzen wollen, um eine Wissensbasis auf dem aktuellen Stand zu halten, dann müssen wir die *Basis* revidieren und die Spezialisierung so gestalten, dass sie die Basis möglichst wenig verändert. Natürlich müssen wir auch die Implikationen bedenken, aber nur in Bezug auf das vom Benutzer oder den bisherigen Lernschritten des Systems gegebene Wissen. Stefan Wrobel (a.a.O.) ersetzt deshalb das erste Gärdenfors-Postulat:

1. $B \frown f \subseteq B \cup \{g' \mid \exists g \in B \text{ so dass } g \geq g'\}$ (minimaler syntaktischer Abstand)

und setzt für B in den Gärdenfors-Postulaten, wo es noch nicht angegeben ist, $Cn(B)$.

Es geht nun darum, einen Revisionsoperator zu konstruieren, der die minimale Basisrevision vornimmt. Für diesen Operator brauchen wir die Buchführung, welche Fakten aus welchen Regeln oder Fakten der Basis abgeleitet werden. Dann kann bestimmt werden, welche Regeln und Fakten der Basis an den auszuschließenden Fakten "beteiligt" waren (Anwendungsmenge). Um die Wiederherstellbarkeit zu gewährleisten, muss ausserdem noch etwas hinzugefügt werden, das uns wieder f ableitet, wenn es denn doch in

[12] Der kurze Beweis in [Wrobel, 1993] zeigt, dass für eine nicht unbedingt abgeschlossene Theorie die maximal generelle Spezialisierung genau dem Ergebnis aller möglichen *maxi-choice contractions* von $Cn(B)$ entspricht.

Ordnung war (Additionsmenge). Das Verfahren von Stefan Wrobel [Wrobel, 1993; Wrobel, 1994] schränkt die Anwendbarkeit einer an der Ableitung von f beteiligten Regel so ein, dass f eben nicht mehr abgeleitet wird. Es fügt eine Regel ein, die mit f als Prämisse das wiederherstellt, was für den Ausschluss von f gelöscht wurde. Da der gesamte Beweisbaum für f berücksichtigt wird und nicht nur der letzte Resolutionsschritt, kann tatsächlich eine minimale Spezialisierung erreicht werden. Dies Verfahren ist einsetzbar für inkrementelles Lernen (engl. online-learning) ebenso wie für die Wartung von Wissensbasen.

1.9 Lernbarkeit in induktiver logischer Programmierung

Wir haben im vorangegangenen Abschnitt 1.7 gesehen, dass auf der Grundlage der Prädikatenlogik das Lernen relationaler Begriffe formalisiert werden kann. Die Frage ist nun, ob diese Formalisierung nur eine theoretische Spielerei ist, oder ob damit auch effiziente Verfahren entwickelt werden können. Die ersten Ergebnisse zur Komplexität [Plotkin, 1971] waren wenig ermutigend: im allgemeinen Fall ist in der Prädikatenlogik erster Stufe nicht entscheidbar, ob die speziellste mit Hintergrundwissen und Beispielen konsistente und gemäß einer Interessantheitsordnung minimale Generalisierung gefunden wird. Inzwischen ist dieser Satz reformuliert worden: die Prädikatenlogik muss eingeschränkt werden, damit eine speziellste Generalisierung von Beispielen (oder eine zu den Beispielen passende generellste Spezialisierung) effizient gefunden werden kann. Um das Lernproblem handhabbar zu machen, werden Beschränkungen der Prädikatenlogik für die Repräsentation der Beispiele (L_E), die des Hintergrundwissens (L_B) und die der Hypothesen (L_H) eingeführt. Zum einen gibt es die Beschränkungen von Hornklauseln.

Definition 1.9.1 (Generative Klausel) *Eine Klausel ist generativ, wenn jede Variable aus dem Klauselkopf auch im Klauselkörper vorkommt. In der Literatur zu deduktiven Datenbanken heissen diese Klauseln bereichsbeschränkt.*

Definition 1.9.2 (Deterministische Klausel) *Eine Klausel ist bezüglich B deterministisch, wenn für jedes Literal im Klauselkörper die bereits durch Konstante substituierten Variablen und das Hintergrundwissen eine eindeutige Substitution für die restlichen Variablen des Literals festlegen. Gibt es mehr als eine Substitution, so ist die Klausel indeterministisch.*

Definition 1.9.3 (Beschränkte Klausel) *Eine Klausel ist beschränkt (constrained), wenn jede Variable aus dem Klauselkörper auch im Klauselkopf vorkommt.*

Solche Beschränkungen erlauben die effiziente Lernbarkeit. Wenn L_B auf eine feste höchste Stelligkeit der Prädikate eingeschränkt wird und L_H aus generativen Klauseln besteht, so können alle Ableitungen in einer Zeit polynomiell zur Anzahl der Klauseln in B durchgeführt werden. Dies bedeutet allerdings noch nicht, dass sie auch polynomiell lernbar sind. Die kombinatorische Explosion des `lgg` kann an folgendem Beispiel deutlich gemacht werden.

$$E^+ = \{\{+(e_1), \quad \neg teil(e_1, a), \neg klein(a), \neg rot(a),$$

$$
\begin{array}{rll}
 & & \neg teil(e_1, b), \neg gross(b), \neg blau(b)\} \\
 & \{+(e_2), & \neg teil(e_2, c), \neg klein(c), \neg blau(c), \\
 & & \neg teil(e_2, d), \neg gross(d), \neg rot(d)\}\} \\
lgg(E^+) = & \{+(E), & \neg teil(E, X_1), \neg klein(X_1), \\
 & & \neg teil(E, X_2), \neg rot(X_2), \\
 & & \neg teil(E, X_3), \neg gross(X_3), \\
 & & \neg teil(E, X_4), \neg blau(X_4)\}
\end{array}
$$

Die zwei Objekte, die in den n Beispielen vorkommen, können ja auf 2^n Arten aufeinander abgebildet werden. Jede Abbildung liefert eine neue existentielle Variable. Es geht also darum, die Anzahl der existentiellen Variablen zu beschränken.

Satz 1.9.1 *Wenn beschränkte Klauseln als L_H gewählt werden, so ist das Lernproblem wahrscheinlich annähernd korrekt lösbar [Jr. und Frisch, 1992].*

Diese Aussage verknüpft das logik-basierte Lernen mit der Theorie des wahrscheinlich annähernd korrekten Lernens (PAC-Lernen, siehe Abschnitt 1.6).

Satz 1.9.2 *Der Aufwand bei deterministischen Klauseln ist exponentiell in der Stelligkeit der Prädikate und der deterministischen Tiefe von Termen. Damit sind sie nicht PAC-lernbar.*

Solche Beschränkungen erlauben die effiziente Lernbarkeit. Wenn L_B auf eine feste höchste Stelligkeit der Prädikate eingeschränkt wird und L_H aus generativen Klauseln besteht, so können alle Ableitungen in einer Zeit polynomiell zur Anzahl der Klauseln in B durchgeführt werden. Wenn beschränkte Klauseln als L_H gewählt werden, so ist das Lernproblem wahrscheinlich annähernd korrekt lösbar [Jr. und Frisch, 1992]. Diese Aussage verknüpft das logik-basierte Lernen mit der Theorie des wahrscheinlich annähernd korrekten Lernens (PAC-Lernen, siehe 1.6). Der Aufwand bei deterministischen Klauseln wächst mit der Stelligkeit der Prädikate und der deterministischen Tiefe von Termen. Nimmt man ein konstantes i für die maximale deterministische Tiefe und ein konstantes j für die maximale Stelligkeit der Prädikate, erhält man ij-deterministische Klauseln. Was ist nun die **deterministische Tiefe** eines Terms?

Definition 1.9.4 (Deterministische Tiefe eines Terms) *Betrachten wir hierzu zunächt die deterministische Verbundenheit von Termen mit dem Klauselkopf.*

Ein Term aus dem Klauselkopf K ist mit einer Kette der Länge 0 deterministisch verbunden.

Für den Klauselkörper nehmen wir an, dass die Literale nach ihrer Verbundenheit mit dem Klauselkopf geordnet sind: $\{\neg L_1, ..., \neg L_m, \neg L_{m+1}, ..., \neg L_n\}$. Ein Term t aus L_{m+1} ist genau dann durch eine deterministische Kette der Länge $d+1$ verbunden, wenn

- *alle Terme im Klauselkopf und in $\{\neg L_1, ..., \neg L_m\}$ verbunden sind durch deterministische Ketten, die höchstens d lang sind, und*
- *es für jede Substitution θ, die K mit einem Beispiel und die ersten Literale mit dem Hintergrundwissen unifiziert – also $K\theta \in E^+$ und $\{\{L_1\}, ..., \{L_m\}\}\theta \subseteq B$ – genau eine eindeutige Substitution σ gibt, so dass $L_{m+1}\theta\sigma \in B$.*

Die deterministische Tiefe eines Terms ist die minimale Länge der deterministischen verbindenden Ketten.

Nimmt man ein konstantes i für die maximale deterministische Tiefe und ein konstantes j für die maximale Stelligkeit der Prädikate, erhält man ij-deterministische Klauseln. Und diese sind polynomiell lernbar. Beschränkte Klauseln sind ein Spezialfall von ij-deterministischen Klauseln ($i = 0$).

Definition 1.9.5 (ij-deterministische Klausel) *Eine Klausel ist* **ij-deterministisch**, *wenn sie*

- *deterministisch ist und*
- *bezüglich B die maximale (deterministische) Tiefe ihrer Terme i ist,*
- *die maximale Stelligkeit ihrer Prädikate j ist.*

Satz 1.9.3 (Lernbarkeit von ij-deterministische Klauseln) *ij-deterministische Klauseln können in polynomieller Zeit gelernt werden, wenn L_E variablenfreie Fakten und L_B variablenfreie Klauseln sind [Muggleton und Feng, 1992].*

Im schlimmsten Falle enthält das Lernergebnis $(Crt)^{ij}$ Literale, wobei C die Anzahl der Prädikate im Hintergrundwissen B, r die höchste Stelligkeit eines Prädikats in B und t die Anzahl von Selektionen im $lgg(E^+)$ ist. Das System GOLEM nimmt feste i und j als Parameter und ist dann polynomiell in der Größe von Hintergrundwissen und Beispielen beschränkt. Die Verbindung zum wahrscheinlich annähernd korrekten Lernen haben Saso Dzeroski, Stephen Muggleton und Stuart Russell hergestellt.

Satz 1.9.4 (PAC-Lernbarkeit von ij-deterministische Klauseln) *Für einfache Verteilungen der positiven und negativen Beispiele sind k funktionsfreie, nicht rekursive ij-deterministische Klauseln wahrscheinlich annähernd korrekt lernbar [Džeroski* et al., *1992].*

L_H ist dann eigentlich nicht ausdrucksstärker als Aussagenlogik. Aber in vielen Fällen ist die Repräsentation dann leichter verständlich.

Eine weitere Beschränkung von Hornklauseln sind k-lokale Klauseln. Sie müssen aber noch weiter differenziert werden, um die Grenze der Lernbarkeit festzulegen.

Definition 1.9.6 (k-lokale Hornklauseln) *Sei D eine Hornklausel*
$D : D_0 \leftarrow D_{DET}, D_{NONDET}$ und vars eine Funktion, die die Menge der Variablen einer Klausel berechnet. $LOC_i \subseteq D_{NONDET}$ ist ein lokaler Teil von D genau dann, wenn $(vars(LOC_i) \backslash vars(\{D_0, D_{DET}\})) \cap vars(D_{NONDET} \backslash LOC_i) = \emptyset$ und es kein $LOC_j \subset LOC_i$ gibt, welches auch ein lokaler Teil von D ist. Ein lokaler Teil LOC_i ist für eine Konstante k:

k-vlokaler Teil *gdw.* $k \geq \mid (vars(LOC) \setminus vars(\{D_0, D_{DET}\}) \mid$,

k-llokaler Teil *gdw.* $k \geq \mid LOC \mid$.

Eine Hornklausel D heisst k-llokal *bzw.* k-vlokal *genau dann, wenn jeder lokale Teil von D ein k-llokaler bzw. k-vlokaler Teil ist.*

Es sind gerade die k-llokalen Klauseln, die polynomiell lernbar sind. Die k-vlokalen Klauseln sind es nicht.

Eine andere Klasse von Formalismen, die die Prädikatenlogik einschränken, sind die terminologischen Logiken oder KL-ONE-Sprachen. Für das Lernen von terminologischen Logiken wurde das Verfahren KLUSTER entwickelt, das in polynomieller Zeit aus einer Menge von variablen- und funktionsfreien Fakten Begriffe lernt, die keine eingebetteten Terme enthalten, also tiefenbeschränkt sind [Kietz und Morik, 1994]. William Cohen und Haym Hirsh zeigten, dass terminologische Logiken wahrscheinlich annähernd korrekt lernbar sind, wenn man den SAME-AS-Operator einschränkt [Cohen und Hirsh, 1992]. Leonard Pitt und Michael Frazer untersuchten die Lernbarkeit der terminologischen Logik CLASSIC, wenn Fragen nach der Klassifikation eines Objekts als Instanz eines Begriffs und Fragen nach der Äquivalenz zweier Begriffe von einem Orakel korrekt beantwortet werden [Frazier und Pitt, 1994].

Den positiven Ergebnissen stehen negative gegenüber. Jörg-Uwe Kietz zeigte, dass die Komplexität des Lernproblems bereits bei nicht tiefenbeschränkten maximal 2-stelligen deterministischen Hornklauseln – d.h. L_H enthält n2-deterministische Klauseln – in PSPACE liegt [Kietz, 1993]. Bei indeterministischen Klauseln nützt selbst die Beschränkung der Tiefe auf 1 und die der Stelligkeit auf 2 nichts: das Problem der Erfüllbarkeit einer Formelmenge (SAT) kann als Lernen von 12-indeterministischen Klauseln formuliert werden. Damit ist das Lernen von 12-indeterministischen Klauseln NP-schwierig [Kietz, 1993]. 12-indeterministische Klauseln sind nicht wahrscheinlich annähernd korrekt lernbar, es sei denn RP=NP. Diese negativen Resultate werden dazu verwendet, den "schlimmen Fall", der dem Beweis zugrunde liegt, abzufangen. Im besten Falle ist der Beweis so geartet, dass der "schlimme Fall" von einem Verfahren erkannt werden kann. Es gibt dann dem Benutzer eine Meldung aus, dass er – mit gedrückten Daumen – den Lernlauf starten mag. Wenn die Lernaufgabe aber als eine polynomiell lösbare vom Verfahren erkannt wurde, löst es diese auch effizient. Auf diese Weise wurde anhand eines Beweises zur Lernbarkeit das bisher schnellste Verfahren zur Subsumtion entwickelt [Kietz und Lübbe, 1994].

1.10 Assoziationsregeln

Der folgende Abschnitt behandelt eine der im Data Mining zurzeit populärsten Analyseaufgaben, nämlich das Entdecken von *Assoziationsregeln* in Datenbanken [Agrawal *et al.*, 1996]. Im Gegensatz zu den in den vorangegangenen Abschnitten behandelten Lernaufgaben ist diese Lernaufgabe keine Spezialisierung des Funktionslernens aus Beispielen. Wir können dies leicht anhand der vielleicht bekanntesten Anwendung von Assoziationsregelverfahren, der sogenannten *Warenkorbanalyse*, erläutern. In den meisten Supermärkten werden zur Beschleunigung des Kassiervorganges und Erleichterung der Bestandsplanung heute Scannerkassen eingesetzt. Ein solches Kassensystem erfasst die auf jeder Verpackung in Form eines Strichkodes aufgedruckte europäische Artikelnummer (EAN), welche jeden Artikel eindeutig identifiziert. Bereits in einem mittleren Supermarkt sind Zehntausende derart unterschiedener Artikel im Angebot, denn es werden nicht nur Hersteller und Produkt, sondern auch unterschiedliche Packungstypen und -größen unterschieden. Anhand der erkannten Artikelnummer kann so zum einen mit Hilfe der im Kassenrechner gespeicherten Preistabelle automatisch der Kassenzettel erstellt werden, zum anderen kann in

einer Datenbank abgelegt werden, aus welchen Artikeln der jeweilige Einkauf (auch *Transaktion* genannt) bestanden hat. Es entstehen so sehr schnell recht große Datenbanken, in denen sich das Kaufverhalten der Kunden widerspiegelt und die darum zur Optimierung der Geschäftsprozesse analysiert werden sollten.

Eine wichtige Frage bei solchen Analysen ist, welche Artikel unter welchen Bedingungen bei einer Transaktion gemeinsam gekauft werden. Interessant könnte also beispielsweise folgende Aussage sein:

> Falls in einem Einkauf Bier und Pizza zusammen gekauft werden, so ist es wahrscheinlich, dass auch Kartoffelchips gekauft werden.

Solche *Assoziationsregeln* können in einem Supermarkt beispielsweise zur Planung der Warenanordnung genutzt werden, so dass also beispielsweise Kartoffelchips in die Nähe von Bier und Pizza platziert werden. Weitere Anwendungen ergeben sich im Versand- und Internethandel. Stellt man beispielsweise fest

> Kunden, die einen Farbdrucker und Bildbearbeitungssoftware bestellen, bestellen wahrscheinlich auch einen Scanner.

so bietet es sich an, denjenigen Kunden, die bisher nur einen Drucker und Bildbearbeitungssoftware gekauft haben, per direktem Brief oder E-Mail ein günstiges Angebot für einen Scanner zu machen.

Wollte man diese Lernaufgabe als Funktionslernen aus Beispielen auffassen, so könnte man versuchen, für jeden Artikel eine Funktion lernen zu lassen, die, gegeben eine partielle Transaktion, vorhersagt, ob der betreffende Artikel ebenfalls in dieser Transaktion vorkommen wird oder nicht. Da wir uns nicht für die Vorhersage eines bestimmten Artikels interessieren, sondern für jede zusätzliche Marketing-Chance, müssten also Zehntausende von Funktionslernaufgaben gelöst und ihre Ergebnisse vom Benutzer analysiert werden. Dies verbietet sich durch den entstehenden hohen Aufwand. Aber selbst wenn wir annehmen, dass wir nur an einem einzigen Artikel interessiert sind, ergibt sich ein weiteres Problem. Es dürfte sich kaum, wie beim Funktionslernen angestrebt, für den gesamten möglichen Instanzenraum aller partiellen Transaktionen verlässlich vorhersagen lassen, ob beispielsweise Kartoffelchips in dieser Transaktionen enthalten sein werden. Dennoch kann es sein, dass für einzelne Bereiche des Instanzenraums eine sinnvolle Aussage möglich ist, z. B. für die Teilmenge der Transaktionen, die Bier und Pizza enthalten. Auch wenn *global* also keine Vorhersage möglich ist, so sind wir doch stark an Aussagen wie der obigen interessiert, welche *lokal* interessante Zusammenhänge *beschreiben*. Solche Lernaufgaben werden deshalb auch als *deskriptive* Lernaufgaben bezeichnet. Da ein beschreibendes Lernverfahren nicht zu einer Aussage über den gesamten Instanzenraum gezwungen ist, kann es seine Suche besser auf das Finden lokal interessanter Zusammenhänge ausrichten. Die Gesamtheit der entdeckten Zusammenhänge ergibt andererseits darum aber nicht notwendigerweise ein globales Modell.

Als eine Spezialisierung der beschreibenden Lernaufgaben können wir das Entdecken von Assoziationsregeln wie folgt präzisieren.

Definition 1.10.1 (Finden von Assoziationsregeln) *Sei I eine Menge von Objekten ("items") und sei T eine Menge von* Transaktionen*, wobei $\forall t \in T : t \subseteq I$. Sei weiterhin $s_{min} \in [0,1]$ eine benutzergegebene Minimalhäufigkeit ("minimal support") und $c_{min} \in [0,1]$*

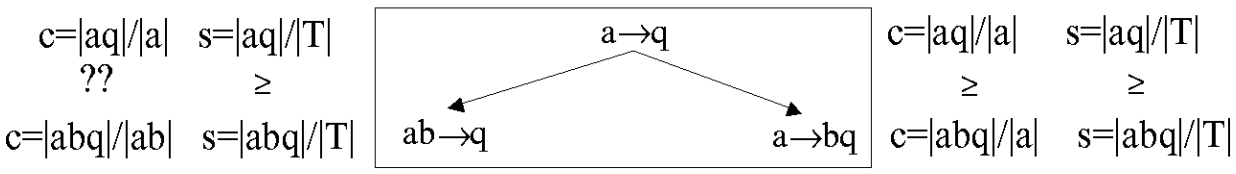

Abbildung 1.11: Absteigende Suche

eine benutzergegebene Minimalkonfidenz. Bei der Lernaufgabe Finden von Assoziationsregeln *sind, gegeben* I, T, s_{min} *und* c_{min}, *alle Regeln der folgenden Form zu finden:*

$X \to Y$

wobei $X \subseteq I$ *und* $Y \subseteq I$ *und* $X \cup Y = \emptyset$, *und es gilt:*

$$s(r) := \frac{\mid \{t \in T \mid X \cup Y \in t\} \mid}{\mid T \mid} \geq s_{min} \quad sowie \quad c(r) := \frac{\mid \{t \in T \mid X \cup Y \in t\} \mid}{\mid \{t \in T \mid X \in t\} \mid} \geq c_{min}$$

In dieser Definition ist also durch zwei Bedingungen spezifiziert worden, wann eine Assoziationsregel als Lösung zulässig ist. Die erste Bedingung verlangt eine gewisse Minimalhäufigkeit s_{min} für die in einer Regel vorkommenden Artikel. Wir sind also nicht an lokalen Zusammenhängen interessiert, die nicht mindestens in einem Anteil von s_{min} aller Transaktionen vorkommt. Die zweite Bedingung spezifiziert, was es bedeutet, dass Artikel "wahrscheinlich" zusammengekauft werden, und verlangt, dass von den Transaktionen, die die Prämisse X beinhalten, mindestens ein Anteil c_{min} auch die Konklusion Y beinhalten soll. Typische Werte sind beispielsweise $s_{min} = 0,01$ und $c_{min} = 0,5$. Finden wir also in der Artikelmenge $I = \{Bier, Pizza, Chips, Wein, ...\}$ mit diesen Einstellungen die Assoziationsregel

$\{Bier, Pizza\} \to \{Chips\}$,

so heißt das, dass mindestens 1 % der Käufer Bier, Pizza und Chips gekauft haben, und dass mindestens 50% der Bier- und Pizza-Käufer auch Chips mitgenommen haben.

1.10.1 Der Apriori-Algorithmus

Verfahren zur Entdeckung von Assoziationsregeln können die Tatsache ausnutzen, dass der Raum aller Assoziationsregeln geordnet werden kann, und dass sich aus dieser Ordnung effiziente Suchstrategien ableiten lassen [Agrawal *et al.*, 1996]. Betrachten wir dazu, was passiert, wenn wir eine Assoziationsregel durch Hinzufügen eines weiteren Objektes in der Prämisse oder der Konklusion verlängern (Abbildung 1.11).

Betrachten wir zunächst in der linken Hälfte der Abbildung, wie sich Häufigkeit und Konfidenz bei Erweiterung der Prämisse verändern. Wie dargestellt, kann die Häufigkeit bei dieser Verfeinerung nur abnehmen, während die Konfidenz sowohl steigen als auch sinken kann, je nachdem, wie sich der hinzugefügte Artikel b auf die Transaktionen mit a und q verteilt. Bei Verlängerung der Konklusion können wir im rechten Teil der Abbildung feststellen, dass sowohl Häufigkeit als auch Konfidenz nur abnehmen können. Dies bedeutet, dass wir bei einer Suche von kürzeren zu längeren Regeln die Suche nur anhand der Häufigkeit beschneiden können: wenn eine Assoziationsregel nicht die nötige Häufigkeit

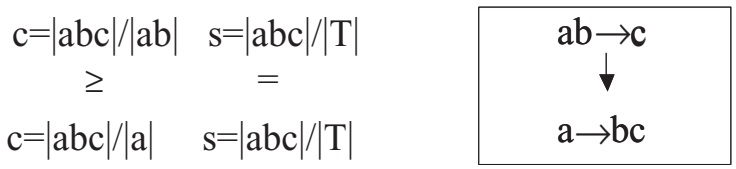

Abbildung 1.12: Regelerzeugung

erreicht, wissen wir, dass dies auch für alle Nachfolger nicht der Fall sein wird. Diese Nachfolger brauchen daher nicht betrachtet zu werden. Wir können in der obigen Abbildung noch eine weitere Beobachtung machen, die für den Entwurf eines Assoziationsregelverfahrens zentral ist. Aus der Definition der Häufigkeit ergibt sich unmittelbar, dass die Verteilung der Artikel auf Prämisse oder Konklusion unerheblich ist. Es kommt nur auf die Gesamtmenge aller in der Regel enthaltenen Artikel an. Wir brauchen also die beiden in der Abbildung dargestellten Zweige nicht zu unterscheiden, sondern können zunächst einfach im Raum der Artikelmengen suchen, von den einelementigen Mengen zu immer größeren Mengen. Wir merken uns all diejenigen Mengen, die die notwendige Mindesthäufigkeit erreichen (die sogenannten *häufigen* Mengen[13]) und schneiden die Suche ab, wann immer wir eine Menge erreichen, die unterhalb der minimalen Häufigkeit liegt.

In einer zweiten Phase müssen dann aus den häufigen Artikelmengen Assoziationsregeln abgeleitet werden (Abbildung 1.12).

Auch dabei können wir wieder die entsprechenden Ordnungseigenschaften ausnutzen. Wie in der Abbildung dargestellt, führt das Bewegen eines Artikels von der Prämisse in die Konklusion einer Regel (beide Regeln entstammen derselben Artikelmenge) dazu, dass die Konfidenz gleich bleibt oder sinkt. Bei der Verwandlung einer Artikelmenge in Assoziationsregeln können wir also mit allen einelementigen Konklusionen beginnen, und dann nach und nach weitere Elemente aus der Prämisse der Konklusion hinzufügen. Zur rechnerischen Auswertung der Konfidenz für eine bestimmte Regel kann übrigens die Tatsache ausgenutzt werden, dass gemäß den Definitionen von c und s: $c(X \rightarrow Y) = \frac{s(XY)}{s(X)}$. Wir können also zur Berechnung der Konfidenz einfach auf die gespeicherten Häufigkeiten aller häufigen Artikelmengen zugreifen[14].

Es ergibt sich der in Abbildung 1.13 dargestellte APRIORI-Algorithmus [Agrawal *et al.*, 1996], bei dem zunächst nur anzumerken ist, dass die notwendige Speicherung der Häufigkeiten $s(c)$ für alle häufigen Mengen c nicht explizit angegeben wurde.

Der Algorithmus enthält jedoch zwei Teilprozeduren, die einer genaueren Betrachtung bedürfen (Abbildung 1.14). Die Teilprozedur PRUNE prüft die Häufigkeit der Mengen und entfernt diejenigen, die die geforderte Minimalhäufigkeit nicht erreichen. Soll dies auf nichthauptspeicherresidenten Daten durchgeführt werden, so ist es günstig, statt einer Schleife, die für jede Kandidatenmenge alle Transaktionen prüft, eine Schleife zu verwenden, die für jede Transaktion alle Kandidatenmengen prüft. So müssen die externen Daten pro Ebene k des Verfahrens nur einmal durchlaufen werden. Um dabei die in einer Transaktion enthalten Kandidaten schnell finden zu können, können *Hashbäume* verwendet werden [Agrawal *et*

[13] Engl. *large* oder *frequent itemsets*.

[14] Da alle Teilmengen einer häufigen Artikelmenge ebenfalls häufig sind, liegt $s(X)$ notwendigerweise ebenfalls vor.

```
proc APRIORI(I, T, s_min, c_min)
    L := HÄUFIGE-MENGEN(I, T, s_min)
    R := REGELN(L, c_min)
    return R
```

```
proc HÄUFIGE-MENGEN(I, T, s_min)
    C_1 := ⋃_{i∈I} {i}, k := 1,
    L_1 := PRUNE(C_1)
    while L_k ≠ ∅
        C_{k+1} := ERZEUGE-KANDIDATEN(L_k)
        L_{k+1} := PRUNE(C_{k+1}, T)
        k := k + 1
    return ⋃_{j=2}^{k} L_j
```

```
proc REGELN(L, c_min)
    R := ∅
    forall l ∈ L, k := |l| ≥ 2
        H_1 := ⋃_{i∈l} {i}, m := 1
        loop
            forall h ∈ H_m
                if s(l_k) / s(l_k \ h) ≥ c_min
                then add l_k \ h → h to R
                else H_m := H_m \ {h}
        while m ≤ k − 2
        H_{m+1} := ERZEUGE-KANDIDATEN(H_m)
        m := m + 1
    return R
```

Abbildung 1.13: APRIORI-Algorithmus

al., 1996].

Die zweite Teilprozedur ERZEUGE-KANDIDATEN erzeugt aus den gefundenen k-elementigen häufigen Mengen alle möglichen $k+1$-elementigen Kandidaten für die nächste Ebene. Statt dabei einfach alle möglichen Artikel an jede bisherige häufige Menge anzuhängen, reicht es aus, alle solchen Paare bisheriger häufiger Mengen zu betrachten, die genau $k-1$ Elemente gemeinsam haben, und diese zu einem dann $k+1$-elementigen Kandidaten zu vereinigen. Dies ist deshalb ausreichend, weil jede Teilmenge einer häufigen Teilmenge ebenfalls häufig sein muss. Gibt es also für eine $k+1$-elementige Menge kein erzeugendes Paar von häufigen Mengen der Größe k, so enthält die Menge offenbar Teilmengen der Größe k, die nicht häufig sind, und kann daher selbst auch nicht häufig sein. Da der Umkehrschluss nicht gilt, wird in der Erzeugungsprozedur jede k-elementige Teilmenge eines Kandidaten

```
proc Erzeuge-Kandidaten(L_k)
    L_{k+1} := ∅
        forall l_1, l_2 ∈ L_k so dass
                    l_1 = {i_1, ..., i_{k-1}, i_k}
                    l_2 = {i_1, ..., i_{k-1}, i'_k}
                    i'_k < i_k (lexikografische Ordnung)
            let l := {i_1, ..., i_{k-1}, i_k, i'_k}
            if alle k-elementigen Teilmengen von l sind in L_k
            then L_{k+1} := L_{k+1} ∪ {l}
    return L_{k+1}
```

```
proc Prune(C_{k+1}, T)
    forall c ∈ C_{k+1} : s(c) := 0
        forall t ∈ T
            forall c ∈ C, c ⊆ t : s(c) := s(c) + 1
    return {c ∈ C | s(c) ≥ s_min · |T|}
```

Abbildung 1.14: APriori-Subroutinen

noch einmal geprüft. Die Kandidatenmengen werden stets lexikografisch sortiert gehalten, um diese Prüfung effizient durchführen zu können. Die Erzeugungsprozedur kann auch bei der Regelgenerierung verwendet werden, denn wie in Abbildung 1.12 dargestellt, gilt für jede Regel mit ausreichender Konfidenz, dass jede Teilmenge ihrer Konklusion ebenfalls zu einer Regel mit ausreichender Konfidenz geführt haben muss (wobei die übrigen Elemente in die Prämisse bewegt werden).

Die Durchführung des Apriori-Algorithmus wird in Abbildung 1.15 anhand eines Beispiels illustriert. In dieser Abbildung sind links die Transaktionen T sowie s_{min} und c_{min} angegeben. In der Mitte finden sich zeilenweise die jeweiligen Kandidatenmengen C_k, ihre Häufigkeiten s und die sich daraus ergebenen häufigen Mengen L_k. Auf der Basis der häufigen Mengen wird im Kasten rechts die Regelerzeugung durchgeführt. Für jede häufige Menge sind die Mengen der ausgewählten Konklusionen A_m angegeben, bei den zweielementigen Mengen ergibt sich ohnehin nur eine Ebene der Suche, bei der dreielementigen Menge $\{1, 2, 3\}$ ist die zweite Ebene leer, weil keine Regel der ersten Ebene eine Lösung war.

Die Performanz des beschriebenen Algorithmus hängt entscheidend von den Eigenschaften der tatsächlich gegebenen Transaktionsmenge T ab. Im allgemeinen Fall können von exponentiell vielen Teilmengen der Artikelmenge I tatsächlich alle häufig sein, mit entsprechenden Folgen für die Laufzeit des Verfahrens. Tatsächlich kauft aber ein Kunde von den 100.000 Artikeln eines Supermarktes nur recht wenige, so dass auch die Größe der häufigen Mengen sehr klein ist und ihre Anzahl mit wachsendem k sehr rasch abnimmt. Sind diese Annahmen gegeben, weist der Algorithmus eine ungefähr lineare Laufzeit in der Größe der Transaktionsmenge T auf. Ebenfalls sehr wichtig für die Gesamtlaufzeit ist dabei aber natürlich die gewählte minimale Häufigkeit. Senkt man die erforderliche minimale Häufigkeit auf die Hälfte, so führte dies in Simulationsexperimenten zu einer Verdoppe-

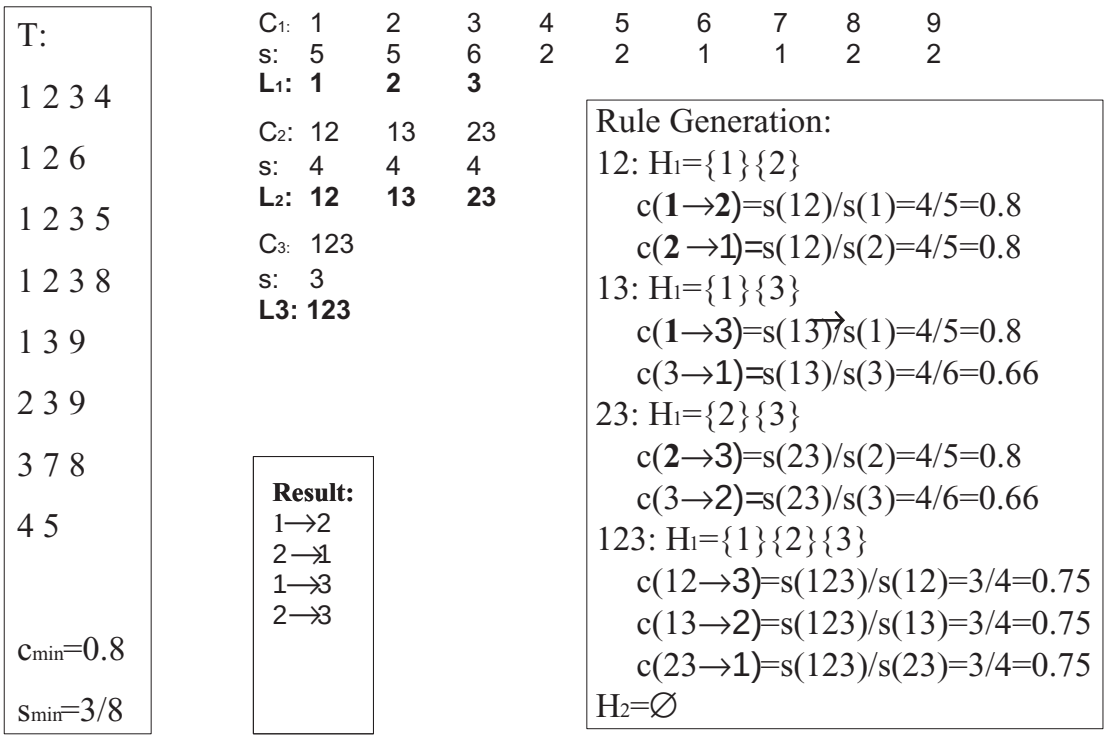

Abbildung 1.15: APriori-Subroutinen

lung der Laufzeit. Die gewählte minimale Konfidenz ist demgegenüber weniger wichtig, da sie nur in der Regelgenerierungsphase benutzt wird, in der ohnehin nur die wenigen gefundenen häufigen Artikelmengen verwendet werden.

1.10.2 Erweiterungen

Auf der Basis des hier beschriebenen Apriori-Algorithmus sind verschiedene Erweiterungen entwickelt worden, die sich einerseits der Verbesserung der Laufzeit des Algorithmus auf großen Datenbeständen widmen, und andererseits den Algorithmus auf weitergehende Aufgabenstellungen erweitern.

Transaktionslisten. Aufgrund der Tatsache, dass jede Transaktion nur wenige der möglichen Artikel enthält, nimmt die Größe der Mengen C_k und L_k mit wachsendem k sehr rasch ab. Dies bedeutet, dass es auch eine wachsende Anzahl von Transaktionen gibt, die keine noch aktuellen Kandidatenmengen mehr enthalten, aber dennoch bei jedem Durchlauf durch die Daten betrachtet werden. Dies kann man verhindern, indem man statt der Transaktion selbst die Liste der in dieser Transaktion noch enthaltenen Kandidatenmengen verwendet, und alle leer gewordenen Listen aus dem Datenbestand entfernt. Für $k = 1$ wäre eine solche Listendarstellung eine 1-zu-1-Kopie der Transaktionsliste, so dass man üblicherweise mit dem Wechsel zur Listendarstellung wartet, bis man davon ausgehen kann, dass diese Darstellung als ganzes in den Hauptspeicher passen wird [Agrawal *et al.*, 1996].

Verallgemeinerte Assoziationsregeln. Bei der Beschreibung des Assoziationsregelverfahrens haben wir bisher die Frage nach der Wahl der richtigen Abstraktionsebene nur implizit beantwortet. Indem wir als Artikelmenge $I = \{Bier, Pizza, Chips, \ldots\}$ angegeben haben, haben wir eine recht abstrakte Ebene gewählt, denn hinter jedem der genannten "Artikel" verbergen sich eigentlich Hunderte von Artikeln unterschiedlicher Hersteller, Produkttypen, Verpackungsarten und -größen. Da es für einen Benutzer nur schwer vorher abzuschätzen ist, ob die interessanten Assoziationen auf der Ebene der Artikelnummer, der Ebene der Hersteller, der Ebene der Produktgruppe oder zwischen diesen Ebenen liegen, sind Verfahren wünschenswert, die Artikelhierarchien (und -mehrfachhierarchien) verarbeiten und Assoziationsregeln auf und zwischen verschiedenen Ebenen dieser Hierarchien finden können. Für solche *verallgemeinerten Assoziationsregeln* gibt es entsprechende Verfahren, die die Eigenschaften der Hierarchie ausnutzen, um ihre Suche möglichst effizient zu gestalten [Srikant und Agrawal, 1995].

Sequentielle Muster. In vielen potenziellen Anwendungsbereichen von Assoziationsregelverfahren ist eine Transaktion nicht so klar definiert wie an der Kasse eines Supermarktes. So kann es bei einem online-Angebot im Internet beispielsweise sein, dass ein Benutzer bei einem Besuch des Angebotes nur eine bestimmte Sache nachfragt, jedoch in mehr oder weniger kurzen Abständen zurückkehrt, um weitere Angebote abzufragen. Zwar könnte man nun festlegen, dass alle innerhalb einer bestimmten Zeitspanne genutzten Angebote zu einer Transaktion gehören, oder dass eine Pause ab einer gewissen Länge den Beginn einer neuen Transaktion markiert. In Verfahren zur Entdeckung von sogenannten *sequentiellen Mustern* kann der Benutzer üblicherweise einen maximalen Abstand zwischen den einzelnen Elementen eines sequentiellen Musters festlegen, und zusätzlich angeben, dass alle Artikel innerhalb eines gleitenden Zeitfensters von vorbestimmter Länge so behandelt werden können, als ob sie in einer Transaktion aufgetreten wären. Ein sequentielles Muster ist eine geordnete Liste von Artikelmengen. Ob ein sequentielles Muster in einer Serie von Transaktionen eines Benutzers vorkommt, wird dann nicht mehr durch einen einfachen Teilmengentest festgestellt, sondern man verlangt, dass zwischen dem frühsten Element einer Artikelmenge in der Liste und dem letzten Element dieser Artikelmenge nicht mehr als die für das gleitende Zeitfenster festgelegte Zeit liegt, dass zwischen zwei aufeinanderfolgenden Artikelmengen in der Liste nicht mehr als der festgelegte Maximalabstand liegt, und dass die Liste eine Subsequenz der tatsächlich aufgetretenen Artikelsequenz ist. Auch für dieses Problem sind effiziente Algorithmen entwickelt worden (siehe z.B. [Srikant und Agrawal, 1996]).

1.11 Subgruppenentdeckung

Wir sind im vergangenen Abschnitt über Assoziationsregelverfahren bereits kurz auf die Besonderheiten von *deskriptiv* orientierten Lernaufgaben im Gegensatz zu *prädiktiv* orientierten Lernaufgaben (Funktionslernen aus Beispielen) eingegangen. Bei prädiktiven Lernaufgaben ist es das Ziel, eine unbekannte Funktion möglichst gut zu approximieren, um so für alle möglichen zukünftigen Instanzen aus dem Instanzenraum den Funktionswert möglichst gut vorhersagen zu können. Hierzu ist also ein *globales* Modell erforderlich,

das es gestattet, für jede mögliche Instanz auch tatsächlich eine Vorhersage zu machen. Hauptziel erfolgreichen Lernens ist das Finden eines Modells mit minimalem wahren Fehler. Beim deskriptiven Lernen hat man andere Ziele. Hier ist man daran interessiert, durch Hypothesen beschriebene Teilbereiche des Instanzenraums zu identifizieren, über die *lokal* interessante Aussagen gemacht werden können.

Betrachten wir zur besseren Illustration ein Anwendungsbeispiel des deskriptiven Lernens. Unser Beispiel soll ein größeres Unternehmen sein, das seinen Kunden mehrere verschiedene Produkte anbietet und eine Kundendatenbank besitzt, in der für jeden Kunden verzeichnet ist, welche Produkte des Unternehmens bereits genutzt werden. Das Unternehmen könnte z. B. eine Versicherung sein, die ihren Kunden Hausrats-, Haftpflicht-, und Lebensversicherungen anbietet. Um sich weitere Marktchancen zu erschließen, wird ein solches Unternehmen daran interessiert sein, Teilbereiche des Instanzenraums, also Gruppen von Kunden, zu identifizieren, in denen beispielsweise ein bestimmtes Produkt auffällig unterrepräsentiert ist. So könnte ein deskriptives Lernverfahren folgende lokale Beobachtungen entdecken:

> "Unter den alleinstehenden jungen Männern in ländlichen Regionen ist der Anteil der Lebensversicherungskunden signifikant niedriger als im gesamten Kundenbestand."

Oder

> "Verheiratete Männer mit Pkws der Luxusklasse machen nur zwei Prozent der Kunden aus, erzeugen aber vierzehn Prozent der Lebensversicherungsabschlusssumme."

Obwohl mit solchen Aussagen keine globale Vorhersage darüber gemacht werden kann, welcher Kunde nun tatsächlich eine Lebensversicherung kaufen wird, sind sie als lokal beschreibende Aussagen dennoch möglicherweise eine wichtige Basis, z. B. für die Planung von Geschäftsstrategien. Mehr noch, in vielen Anwendungen lassen sich solche interessanten lokalen Aussagen finden, obwohl ein globales prädiktives Modell (z. B. aufgrund nicht genügend reichhaltiger Beschreibung der Instanzen) nicht oder nur mit sehr schlechtem Erfolg erlernt werden kann.

Lernaufgaben wie die gerade beschriebenen werden üblicherweise als *Subgruppenentdeckung* [Klösgen, 2000c] bezeichnet, da man, in Anlehnung an die in der Statistik übliche Terminologie, den Instanzenraum als (Repräsentation einer) *Population* bezeichnen kann, so dass ein beschriebener Teilbereich des Instanzenraum dann als Subgruppe dieser Population aufgefasst wird. Wir können die Lernaufgabe wie folgt präzisieren.

Definition 1.11.1 (Subgruppenentdeckung) *Sei X ein Instanzenraum mit einer Wahrscheinlichkeitsverteilung D und L_H ein Hypothesenraum, in dem jede Hypothese als* Extension *eine Teilmenge von X hat:*

$$ext(h) \subseteq X \text{ für alle } h \in L_H.$$

Sei weiterhin

$$S \subseteq X$$

eine gegebene, gemäß D gezogene Stichprobe der Gesamtpopulation.

Es sei schließlich q eine Funktion

$$q := L_H \to \mathbb{R}.$$

Die Lernaufgabe Subgruppenentdeckung *kann dann auf zwei Arten definiert werden:*

1. *Gegeben X, S, L_H, q und eine Zahl $q_{min} \in \mathbb{R}$, finde alle $h \in L_H$, für die $q(h) \geq q_{min}$. Oder/Und*
2. *gegeben X, S, L_H, q und eine natürliche Zahl $k \geq 1$, finde eine Menge $H \subseteq L_H, |H| = k$ und es gibt keine $h \in H, h' \in L_H \backslash H : q(h') \geq q(h)$.*

Eine zentrale Rolle in dieser Definition spielt die Qualitätsfunktion q, die bewertet, wie "interessant" eine bestimmte Hypothese bzw. die durch sie repräsentierte Subgruppe $ext(h)$ ist. Gegeben q verlangen wir bei der Subgruppenentdeckung entweder alle Subgruppen oberhalb einer bestimmten Mindestqualität q_{min}, oder/und die k gemäß q besten Hypothesen aus unserem Hypothesenraum.

1.11.1 Qualitätsfunktionen

Wie kann nun die Funktion q aussehen, welche die Interessantheit von Aussagen messen soll? Um die tatsächliche Interessantheit einer Beobachtung für einen Benutzer zu messen, wäre eigentlich eine umfassende Benutzermodellierung notwendig, die Ziele, Interesse und Vorwissen des Benutzers erfasst. So sind beispielsweise bereits bekannte Aussagen für die meisten Benutzer nicht besonders interessant. Auch wenn es erste Arbeiten zur Entwicklung von präzisen benutzerbezogenen Interessantheitsmodellen gibt, so ist es momentan doch noch allgemein üblich, die Interessantheit von Aussagen anhand leicht objektivierbarer statistischer Eigenschaften der betrachteten Subgruppe zu messen [Klösgen, 2000b]. Üblicherweise werden dabei zwei unterschiedliche Gesichtspunkte einbezogen. Wie in den obigen Beispielaussagen schon angedeutet, interessiert man sich zunächst einmal für Subgruppen, die statistisch auffällig sind, weil beispielsweise die Verteilung eines bestimmten Merkmals von Interesse ("abgeschlossene Lebensversicherungen") sich "signifikant" von der Verteilung in der gesamten Stichprobe bzw. der gesamten Population unterscheidet. Als zweiter Gesichtspunkt muss aber die Größe der betrachteten Subgruppe einbezogen werden, denn auch eine bemerkenswerte statistische Auffälligkeit ist nur dann interessant, wenn sie mehr als einige wenige Objekte aus der Population betrifft.

Eine naheliegende Möglichkeit, den Begriff "statistisch auffällig" zu präzisieren, ist der Rückgriff auf das Instrumentarium statistischer Hypothesentests (siehe z.B. [Diehl und Arbinger, 1990]). Betrachten wir dazu die einfachste Variante eines Subgruppenentdeckungsproblems und nehmen an, dass wir an Subgruppen interessiert sind, die hinsichtlich der Verteilung eines binären Attributs von Interesse auffällig sind. Im ersten Beispiel dieses Abschnitts war dies das Attribut "Lebensversicherung abgeschlossen?". Wir können uns nun ein Experiment vorstellen, bei dem wir (gemäß der Verteilung D) zufällig ein Objekt aus der Gesamtpopulation ziehen und prüfen, ob die gesuchte Eigenschaft bei diesem Objekt vorliegt oder nicht (ob also der Kunde eine Lebensversicherung abgeschlossen hat oder nicht). Die Verteilung der gesuchten Eigenschaft in der Gesamtpopulation entspricht einer Wahrscheinlichkeit p, bei einer Ausführung unseres Experiments ein Objekt mit der gewünschten Eigenschaft zu ziehen.

Um nun die statistische Auffälligkeit einer durch eine Hypothese h beschriebenen Subgruppe der Größe $n := |ext(h)|$ zu bewerten, betrachten wir die Wahrscheinlichkeit p_h, bei

Ziehung eines Objektes aus dieser Subgruppe ein Objekt mit der gesuchten Eigenschaft zu erhalten. Als statistisch hinsichtlich der gesuchten Eigenschaft nicht auffällig wollen wir nun jede Subgruppe h betrachten, die sich hinsichtlich des gesuchten Attributs genau so verhält, wie eine zufällig aus der Gesamtpopulation ausgewählte Stichprobe der Größe n. Dies ist unsere durch einen geeigneten statistischen Test möglicherweise zu verwerfende *Nullhypothese* über die Subgruppe h. Gilt die Nullhypothese, so ist $p_h = p$, und das Ziehen einer Subgruppe der Größe n ist die n-fache Wiederholung eines Experiments, bei dem mit der Wahrscheinlichkeit p ein Objekt mit der gesuchten Eigenschaft gezogen wird. Die relative Häufigkeit solcher Objekte in dieser Stichprobe, also in $ext(h)$, ist dann also ein geeigneter Schätzer für p:

$$\hat{p} := \frac{|\{s \in ext(h) | A(s) = 1\}|}{n}$$

Dieser Schätzer ist *erwartungstreu*, d. h., wenn wir wiederholt Stichproben der Größe n ziehen, wird der Wert dieses Schätzers gemittelt über alle Stichproben schließlich der zu Grunde liegenden Wahrscheinlichkeit entsprechen, also unter der Annahme der Nullhypothese, dass $p_h = p$:

$$E(\hat{p}) = p.$$

In den einzelnen Stichproben der Größe n schwankt der Wert unseres Schätzers je nach den für die jeweilige Stichprobe ausgewählten Objekte. Was können wir aus dem in einer einzelnen Stichprobe der Größe n, nämlich unserer Subgruppe h, beobachteten Wert von $\hat{p}$ über die Gültigkeit der Nullhypothese schließen? Betrachten wir dazu die Wahrscheinlichkeit, mit der ein bestimmter Wert von $\hat{p}$ auftritt, falls p tatsächlich die zu Grunde liegende Wahrscheinlichkeit der gesuchten Eigenschaft ist. Je unwahrscheinlicher ein beobachteter Wert ist, desto weniger plausibel ist es, dass tatsächlich p die zugrundeliegende Wahrscheinlichkeit war, und desto naheliegender ist es, die Nullhypothese zu verwerfen, und anzunehmen, dass die betrachtete Subgruppe sich tatsächlich von der Gesamtpopulation statistisch unterscheidet. Betrachten wir als Beispiel eine Population, in der ein gesuchtes Merkmal mit der Wahrscheinlichkeit $p = 0,5$ auftritt. Bei Ziehung einer Subgruppe der Größe $n = 2$ erwarten wir im Mittel ein Objekt mit der gesuchten Eigenschaft; die Wahrscheinlichkeit, zwei Objekte mit dieser Eigenschaft anzutreffen, ist jedoch immer noch $P(\hat{p} = 1 | p = 0,5, n = 2) = 0,5 \cdot 0,5 = 0,25$.

Wenn wir in einer Subgruppe der Größe 2 diese Verteilung beobachten, ist dies also kein besonders starker Hinweis darauf, dass die Subgruppe sich von der Gesamtpopulation unterscheidet. Die sich für allgemeines n ergebene Wahrscheinlichkeitsverteilung $P(\hat{p} = x | p, n)$ der Werte von $\hat{p}$ (genauer die Verteilung der Anzahl der gezogenen Objekte mit der gesuchten Eigenschaft) ist die *Binomialverteilung*. Für genügend große Werte von n ($n > 30$ wird allgemein akzeptiert) wird diese Verteilung genügend genau durch eine Normalverteilung mit Mittelwert $\mu = p$ und Standardabweichung $\sigma = \sqrt{\frac{p \cdot (1-p)}{n}}$ genähert.

Man könnte nun direkt die Wahrscheinlichkeit $P(\hat{p} = x | p, n)$ eines beobachteten Wertes x für $\hat{p}$ als Maß für die Plausibilität einer Hypothese nehmen, müsste allerdings dann für verschiedene p und n unterschiedliche Verteilungstabellen nutzen. Man führt daher eine so genannte z-Transformation durch, die aus einer beliebigen normalverteilten Variablen

eine standard-normalverteilte Variable ($\mu = 0$ und $\sigma = 1$) erzeugt. Dies ergibt hier die V genannte Variable

$$V := \frac{\hat{p} - p}{\sqrt{\frac{p \cdot (1-p)}{n}}}.$$

Der V-Wert hat auch die intuitiv günstige Eigenschaft, daß Abweichungen als Vielfaches der Standardabweichung ausgedrückt werden, so daß die für kleinere n stärker schwankenden Werte von $\hat{p}$ günstiger skaliert sind. Als Maß für die Plausibilität der Nullhypothese wählen wir dann $P(|V| \geq V)$, also die Wahrscheinlichkeit, dass die skalierte Abweichung unseres Schätzers von p mindestens v beträgt[15]. Die entsprechenden Werte können den Tabellen in Statistikbüchern entnommen werden, so ist z. B. die Wahrscheinlichkeit, eine skalierte Abweichung von mehr als 4,9 zu beobachten nur $\frac{1}{1000000}$. Beobachtet man einen solchen Wert von V bei einer Subgruppe, so ist das also ein sehr starkes Indiz dafür, dass die Nullhypothese nicht gilt, und dass also die Subgruppe eine andere Verteilung des gesuchten Merkmals aufweist als die Gesamtpopulation.

Für die Definition einer entsprechenden Qualitätsfunktion q können wir noch einige Vereinfachungen vornehmen, denn wir sind in der Regel ja nur an der Reihenfolge interessiert, die q den Hypothesen zuweist und nicht am konkreten absoluten Wert. Da $P(|V| \geq v)$ für wachsende v monoton fallend ist, und wir ohnehin kleineren Wahrscheinlichkeiten eine höhere Qualität (Auffälligkeit) zu ordnen wollen, führt die Verwendung des Wertes von V statt der Wahrscheinlichkeit zu einer wie erwünscht genau umgekehrten Reihenfolge der Hypothesen. Da die Bezugswahrscheinlichkeit p der Gesamtpopulation für alle zu bewertenden Subgruppen gleich ist, können wir wiederum ohne Änderung der Reihenfolge wie folgt umformen:

$$|V| := \frac{|\hat{p} - p|}{\sqrt{\frac{p \cdot (1-p)}{n}}} = \frac{\sqrt{n} \cdot |\hat{p} - p|}{\sqrt{p \cdot (1-p)}} \sim \sqrt{n} \cdot |\hat{p} - p| \sim \sqrt{g} \cdot |\hat{p} - p|.$$

Im letzten Schritt dieser Umformung ist die Größe der Subgruppe n ersetzt worden durch ihre relative Größe $g := \frac{n}{|S|}$. Um diese Qualitätsfunktion evaluieren zu können, muss natürlich die Wahrscheinlichkeit p für die Gesamtpopulation geschätzt werden. Dies tut man in nahe liegender Weise durch die relative Häufigkeit der gesuchten Eigenschaft in S:

$$\hat{p}' := \frac{|\{s \in S | A(s) = 1\}|}{|S|}.$$

Wir erhalten also die Qualitätsfunktion[16]

$$q(h) := \sqrt{g} \cdot |\hat{p} - \hat{p}'|.$$

Die Qualitätsfunktion q, so wie wir sie gerade definiert haben, ordnet also die Subgruppen nach absteigender Signifikanz an. Die besten k Subgruppen dieser Liste müssen also

[15] Dies ist der zweiseitige Test. Sind nur Abweichung in eine Richtung von Interesse, so entfallen die Betragsstriche.

[16] In Artikeln über Subgruppenentdeckung ist es üblich, statt $\hat{p}'$ p_0 zu verwenden, und für $\hat{p}$ einfach p.

keineswegs Subgruppen sein, bei denen man die Nullhypothese mit genügender Sicherheit ausschließen kann. Um dies sicherzustellen, müssen wir ein Mindestqualitätsniveau q_min vorgeben, das sich gemäß den obigen Umformungen aus dem gewünschten Signifikanzniveau des statistischen Hypothesentests ergibt. Dabei tritt jedoch noch eine weitere Schwierigkeit auf. Wenn wir bei jeder einzelnen Hypothese mit 95%-iger Sicherheit ausschließen können, dass die betrachtete Subgruppe fehlerhafterweise als auffällig erkannt wurde, so bedeutet das immer noch, dass man bei 5% aller getesteten Subgruppen eine fehlerhaft erkannte Auffälligkeit erwarten muss. Dieses Problem kann dadurch entschärft werden, dass man das Signifikanzniveau für den einzelnen Test entsprechend der Größe des Hypothesenraums verschärft (Bonferroni-Anpassung).

1.11.2 Effiziente Suche

Verfahren zur Entdeckung von Subgruppen müssen gemäß der Definition der Lernaufgabe nicht nur eine Lösung finden, sondern die Menge aller Lösungen oberhalb eines bestimmten Qualitätsniveaus bzw. die Menge der k besten Lösungen im Hypothesenraum. Die Verfahren können deshalb auch keine Hill-climbing-Suche durchführen, sondern müssen versuchen, den Hypothesenraum möglichst vollständig zu durchsuchen. Dies kann im einfachsten Fall z. B. mit einer absteigenden Breitensuche in L_H geschehen (Abbildung 1.16). Man beginnt also mit der allgemeinsten Hypothese in diesem Raum, und verfeinert diese dann zu immer kleineren Subgruppen. Schon bei einfachen Problemen, bei denen die Instanzen nur durch wenige Merkmale beschrieben sind, ergibt sich jedoch ein so großer Hypothesen- und damit Suchraum, dass es notwendig ist, Kriterien zu finden, mit denen möglichst große Teile des Suchraumes abgeschnitten und damit von der Betrachtung ausgeschlossen werden können.

Am leichtesten ist dies möglich, wenn man für die Problemstellung relevante Eigenschaften der Hypothesen finden kann, die sich entlang eines absteigenden Suchpfades monoton entwickeln, also z. B. niemals größer, sondern nur kleiner werden können. Sobald bei einer absteigenden Suche diese Größe einen geforderten Mindestwert unterschreitet, können alle darunterliegenden Hypothesen als Lösung ausgeschlossen werden. Für die bei der Subgruppenentdeckung relevante Eigenschaft, die Qualität q, gilt leider eine solche Monotonieeigenschaft im allgemeinen Fall nicht. Je nachdem, wie sich die Verteilungen bei einer Verkleinerung der Subgruppe ändern, kann die Qualität der Subgruppe fallen oder auch steigen.

Für die zweite Variante des Subgruppenentdeckungsproblems, das Finden der k besten Subgruppen, kann man dennoch erfreulicherweise in vielen Fällen große Teile des Suchraumes abschneiden, indem man sich sogenannter *optimistischer Schätzfunktionen* für q bedient [Wrobel, 1997]. Das Ziel einer optimistischen Schätzfunktion ist es, gegeben eine bestimmte Hypothese h und ihre Qualität $q(h)$, abzuschätzen, wie sich die Qualitätsfunktion q bei einer weiteren absteigenden Suche ausgehend von h günstigstenfalls entwickeln könnte. Die optimistische Schätzfunktion sollte garantieren, dass für keine der unterhalb von h liegenden spezielleren Hypothesen der Wert von q oberhalb der optimistischen Schätzung liegt. Können wir dies zusichern, so kann man immer dann den Teilraum unterhalb einer Hypothese h abschneiden, wenn die optimistische Schätzung für diesen Teilraum unterhalb

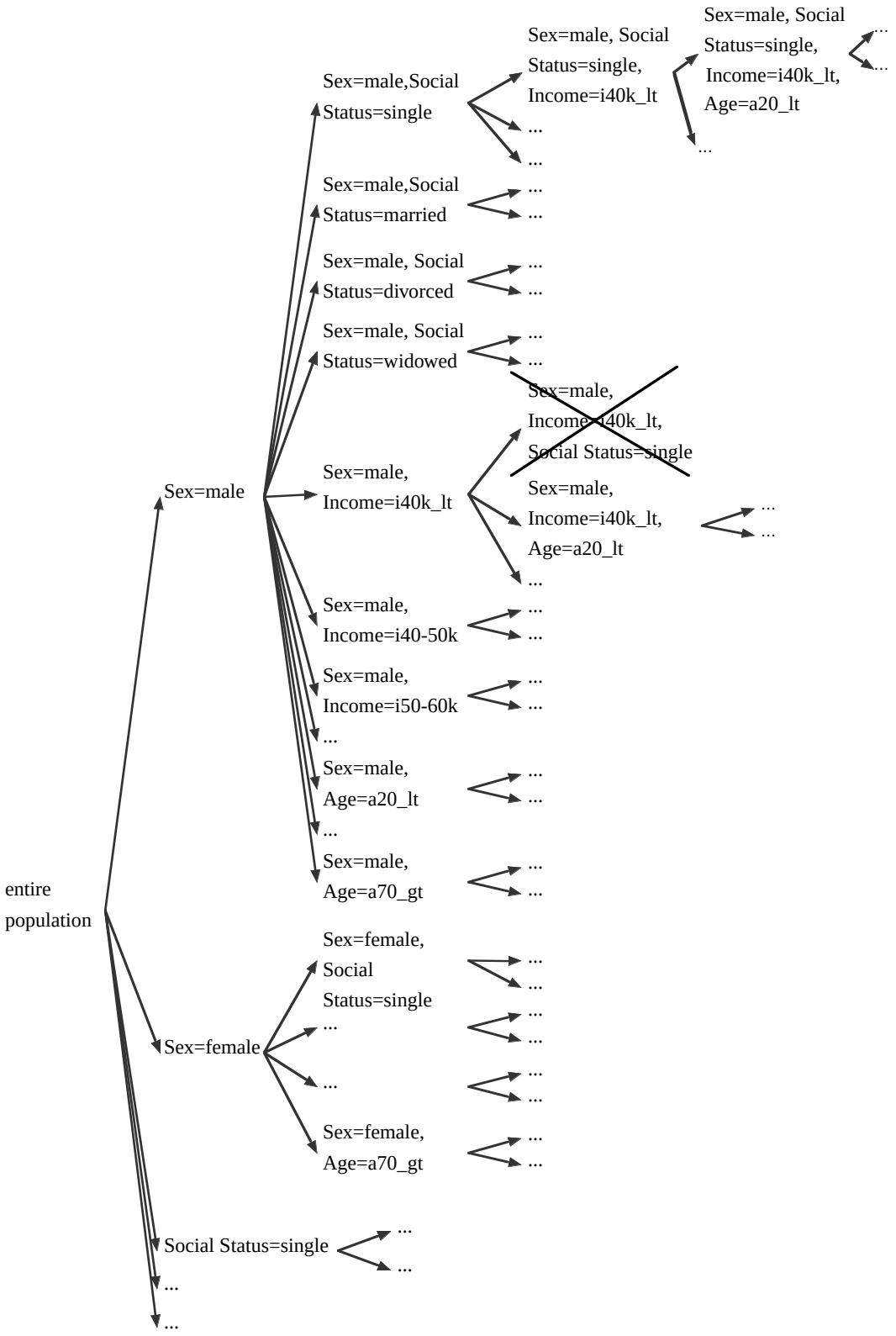

Abbildung 1.16: Suchbaum bei der propositionalen Subgruppensuche

```
proc SUBGRUPPEN(S, ρ, q, q_min, g_min, k)
    Q := {"Ganze Population"}, H := ∅
    while Q ≠ ∅
        Wähle C ⊆ Q (gemäß Suchstrategie)
        ρ(C) := ⋃_{h∈c} ρ(h)
        forall h ∈ ρ(C)
            Teste h auf S (berechne q(h))
            if q(h) ≥ q_min und g(h) ≥ g_min then
                if q(h) > min_{h∈H} q(h) then
                    if |H| < k
                        then H := H ∪ {h}
                        else ersetze schlechtestes Element von H durch h
            if q_max(h) < min_{h∈H} q(h) or g(h) < g_min
                then (Teilbaum wird abgeschnitten)
                else Q := Q ∪ {h}
    return H
```

Abbildung 1.17: Subgruppensuche mit optimistischen Schätzfunktionen

der Qualität der schlechtesten der bisher gefundenen k Hypothesen liegt.

Wir können dies wie folgt präzisieren. Sei $\rho : L_H \to 2^{L_H}$ der Spezialisierungsoperator, der uns bei der absteigenden Suche zu einer Hypothese h alle unmittelbar spezielleren Nachfolger $\rho(h)$ liefert, und sei ρ^* die transitive Hülle von ρ, also der gesamte Teilraum unterhalb einer Hypothese h. Eine optimistische Schätzfunktion für q ist dann eine Funktion:

$$q_{max} : L_H \to \mathbb{R}, \text{so dass } \forall h \in L_H, \forall h' \in \rho(h)\ q(h') \leq q_{max}(h)$$

Mit Hilfe einer solchen optimistischen Schätzfunktion kann ein Suchgruppenentdeckungsalgorithmus wie in Abbildung 1.17 formuliert werden. In diesem Algorithmus wird optional noch eine vom Benutzer angegebene minimale Größe der zu entdeckenden Subgruppen zum Beschneiden des Suchraumes genutzt.

Je nach Beschaffenheit der Qualitätsfunktion q ist es unterschiedlich schwierig, eine optimistische Schätzfunktion anzugeben, bzw. nachzuweisen, dass sie überhaupt existiert. Wir wollen im folgenden eine optimistische Schätzfunktion für den einfachsten Fall, der Suche nach Subgruppen in Bezug auf ein binäres Zielattribut, angeben, und zwar für die oben erläuterte Qualitätsfunktion

$$q(h) = \sqrt{g} \cdot |\hat{p} - \hat{p}'|.$$

Da g, die Größe der Subgruppe, bei der Spezialisierung nur kleiner werden kann, können wir den ersten Faktor einfach optimistisch mit dem gleichen Wert wie in der aktuellen Hypothese h schätzen. Im zweiten Faktor kann sich $\hat{p}$ im Intervall $[0, 1]$ sowohl nach oben als auch nach unten bewegen. Eine erste grobe optimistische Schätzfunktion, mit der wir uns an dieser Stelle begnügen wollen, ist deshalb

$$q_{max}(h) := \sqrt{g} \cdot max(\hat{p}', 1 - \hat{p}').$$

1.11.3 Assoziationsregeln vs. Subgruppen

Es sei zum Schluss noch darauf hingewiesen, dass man auch das Entdecken von Assoziationsregeln als Instanz der Lernaufgabe Subgruppenentdeckung betrachten kann. Jede Assoziationsregel repräsentiert die Subgruppe all derjenigen Transaktionen, in denen alle Artikel der Assoziationsregel vorkommen. Man kann dann eine Qualitätsfunktion q definieren, die für eine Assoziationsregel r den Wert 1 hat genau dann, wenn $s(r) \geq s_{min}, c(r) \geq c_{min}$, und ansonsten den Wert 0. Auffälligkeit wird also nicht durch Vergleich mit einer Referenzgruppe definiert, sondern einfach durch einen minimalen Prozentsatz gemeinsamen Auftretens. Die minimale Häufigkeit kommt einfach als zweite Minimalbedingung hinzu und wird nicht mit der Auffälligkeit verrechnet, und es findet keine Differenzierung unterschiedlicher Interessantheitsgrade statt.

Auch wenn man also formal die Assoziationsregeln als Subgruppenentdeckung sehen kann, so ist es aus den gerade genannten Gründen technisch nicht sinnvoll, dies zu tun. Die extrem simple Qualitätsfunktion und die im Abschnitt über Assoziationsregeln diskutierten besonderen Annahmen über die Zusammensetzung der Transaktionen ermöglichen bzw. erfordern ganz andere Suchverfahren. Insbesondere gelten bei den üblicherweise verwendeten mächtigeren Qualitätsfunktionen zur Subgruppenentdeckung die in Abbildung 1.11 gemachten Beobachtungen nicht, denn die Qualität kann bei absteigender Suche sowohl fallen als auch sinken, so dass q, wie oben beschrieben, nicht direkt zur Beschneidung des Suchraums genutzt werden kann.

1.11.4 Erweiterungen

Die Weiterentwicklung von Subgruppenverfahren erfolgt entlang mehrerer Dimensionen. Wie bereits erwähnt sind für unterschiedliche Anwendungszwecke vielfältige Qualitätsfunktionen entwickelt worden [Klösgen, 2000b]. Weitere Entwicklungen betreffen die Nutzung mächtigerer Hypothesenräume, die Einführung zeitbezogener Subgruppenmuster, und die Optimierung von Subgruppenergebnissen.

Mächtigere Hypothesenräume. Die in diesem Abschnitt beschriebene Subgruppensuche kann selbstverständlich nicht nur für Hypothesenräume durchgeführt werden, die sich auf Merkmalsvektoren beziehen, sondern für jeden Hypothesenraum, in dem sich ein Verfeinerungsoperator ρ definieren lässt. So gibt es beispielsweise Subgruppenverfahren für multirelationale prädikatenlogische Hypothesenräume, bei denen durch geeignete Sprachbeschränkungen sichergestellt wird, dass der entstehende Suchraum handhabbar bleibt. Neben den im Abschnitt 1.7 diskutierten syntaktischen Schemata haben sich hier Einschränkungen bewährt, die an die Fremdschlüsselbeziehungen in Datenbanken angelehnt sind [Wrobel, 1997].

Zeitbezogene Subgruppenmuster. Bei vielen Marktuntersuchungen ist es üblich, die Untersuchungen in bestimmten Abständen zu wiederholen. So könnte beispielsweise ein Unternehmen eine jährliche Kundenbefragung durchführen und daran interessiert sein, die Gesamtheit der Fragebogenrückläufe auch im zeitlichen Trend mit Subgruppenverfahren zu untersuchen. Es sind zu diesem Zweck spezielle Qualitätsfunktionen entwickelt worden, die nicht nur einen Vergleich von Subgruppen mit einer Gesamtpopulation durchführen,

sondern dies auch über mehrere in zeitlichen Abständen erhobene Teilpopulationen ("Segmente") tun können [Klösgen, 2000a]. Dabei muss als technische Schwierigkeit beachtet werden, dass die Beschreibungen der Instanzen in den einzelnen Segmenten nicht notwendigerweise die gleichen Attribute aufweisen [Klösgen, 1996].

Optimierung von Subgruppenergebnissen. Formuliert man deskriptive Lernprobleme mit Qualitätsfunktionen, die auf die einzelnen Hypothesen bezogen sind, so ergibt sich zwangsläufig das Problem, dass die Gesamtmenge aller gefundenen Hypothesen möglicherweise in sich nicht gut aufeinander abgestimmt ist. So kann es bei der Subgruppenentdeckung ohne weitere Vorkehrungen beispielsweise vorkommen, dass wir entdecken, dass alleinstehende Männer doppelt so häufig Lebensversicherungen abschließen wie die Gesamtpopulation, und dass wir gleichzeitig als Entdeckung zurückliefern, dass alleinstehende Männer in ländlichen Gegenden doppelt so häufig Lebensversicherungen abschließen wie die Gesamtbevölkerung. Allgemein ergibt sich also die Frage, wie inhaltlich ähnliche Subgruppen erkannt und dann die weniger interessante von zwei sehr ähnlichen Subgruppen unterdrückt werden kann. In unserem Beispiel ist vermutlich die zweite Beobachtung weniger interessant, weil ihr Inhalt vollständig aus der ersten Beobachtung ableitbar ist. Man kann zur Unterdrückung solch redundanter Subgruppen Maße definieren, die die sogenannte *Affinität* von Subgruppen berechnen, und kann dann Schwellwerte festlegen, so dass oberhalb einer gewissen Affinität und unterhalb eines bestimmten Unterschieds in der Interessantheit die schwächere der beiden Subgruppen unterdrückt wird [Gebhardt, 1991].

1.12 Clusteranalyse

Bei der Clusteranalyse [Steinhausen und Langer, 1977; Bacher, 1996] geht es grob gesagt darum, eine gegebene Instanzenmenge S aus einem Instanzenraum X so in verschiedene Teilmengen ("Cluster") aufzuteilen, dass die Objekte innerhalb eines Clusters möglichst ähnlich sind, und dass Objekte verschiedener Cluster einander möglichst unähnlich sind. Diese Lernaufgabe findet immer dann Anwendung, wenn man statt einer großen Zahl einzelner Instanzen lieber mit einer überschaubar kleinen Anzahl idealerweise homogener Gruppen umgehen möchte. So könnte man beispielsweise die Kundendatenbank eines Unternehmens einer Clusteranalyse unterziehen, um homogene Kundengruppen zu finden, mit denen dann in der Angebotsgestaltung und im Marketing gearbeitet werden kann. Im Idealfall sollte ein Clusteringverfahren dabei nicht nur die gefundenen Teilmengen mit den Listen der zu ihnen gehörenden Instanzen liefern, sondern auch eine kompakte und möglichst leicht verständliche Beschreibung für jeden Cluster, die die in dem Cluster enthaltenen Objekte charakterisiert. So könnte eine Clusteranalyse einer Kundendatenbank beispielsweise die Cluster "auf dem Land lebende Menschen mit Kindern", "Senioren mit hohem Einkommen", "beamtete Akademiker" usw. ergeben. Man spricht in diesem Fall auch von *begrifflicher Clusteranalyse* (*conceptual clustering*) [Michalski und Stepp, 1983]. In den meisten Anwendungen verlangt man, dass die gefundenen Cluster sich nicht überlappen sollen, und dass jede mögliche Instanz des Instanzenraumes X einem Cluster zugeordnet werden kann, d.h., die Menge aller Cluster soll eine *Partitionierung* des Instanzenraumes darstellen.

Im Vergleich zur Subgruppenentdeckung ist hervorzuheben, dass die beim Clustern gefundenen Gruppen nicht hinsichtlich ihrer Eigenschaften in Bezug auf ausgewählte Zielmerkmale identifiziert werden, sondern ausschließlich durch die Forderung nach einer maximalen Ähnlichkeit der Instanzen innerhalb eines Clusters. Die Ähnlichkeits- bzw. Abstandsfunktion nimmt dementsprechend einen wichtigen Platz in der präzisen Definition der Lernaufgabe ein.

Definition 1.12.1 (Cluster-Analyse) *Sei X ein Instanzenraum und $S \subseteq X$ eine Menge von Instanzen. Sei weiterhin*

$$dist : X \times X \to \mathbb{R}^+$$

eine Abstandsfunktion, und

$$q : 2^{2^X} \to \mathbb{R}$$

eine Qualitätsfunktion. *Gegeben S, dist und q besteht die Aufgabe der* Clusteranalyse *darin, eine* Clusterung

$$\mathcal{C} = \{C_1, \ldots, C_k\},\ \textit{wobei}\ C_i \subseteq S\ \textit{für alle}\ i = 1, \ldots, k,$$

zu finden, so dass $q(\mathcal{C})$ maximiert wird, und (optional)

$$C_i \cap C_j = \emptyset\ \textit{für alle}\ i \neq j \in \{1, \ldots k\}\ \textit{und}\ \bigcup_{i=1,\ldots,k} C_i = \mathcal{S}\ \textit{(Partitionierung).}$$

Bezüglich der Ähnlichkeitsfunktionen können wir an dieser Stelle einfach auf die Ausführungen im Abschnitt 1.4 verweisen. Die Qualitätsfunktion q wird oft auf der Basis der Abstandsfunktion *dist* definiert, indem man beispielsweise für jeden Cluster den durchschnittlichen Abstand aller Objektpaare in diesem Cluster berechnet, und die negierte Summe dieser Werte für alle Cluster als Qualitätsmaß nimmt. Ist es in einem Instanzenraum X möglich, das Zentrum eines Clusters zu bestimmen, so wird statt des durchschnittlichen paarweisen Abstandes auch der durchschnittliche (quadrierte) Abstand (Fehler) vom Zentrum verwendet (s.u. beim k-Means-Verfahren). Ist die Anzahl der gewünschten Cluster k nicht vom Benutzer vorgegeben, so ist es wichtig, dass die Qualitätsfunktion die Anzahl der Cluster und ihre innere Qualität gegeneinander abwägt. Mittels der Qualitätsfunktion kann auch die gewünschte Zuordnung neuer Instanzen aus X zu einem Cluster gefunden werden: man weist die Instanz einem Cluster so zu, dass danach die Gesamtqualität des Clusterings maximal wird.

1.12.1 Das k-Means-Verfahren

Ein sehr einfaches und populäres Verfahren zur Suche nach Clusterungen der Größe k für vorgegebenes k ist das k-Means-Verfahren [MacQueen, 1967]. Das k-Means-Verfahren setzt voraus, dass es möglich ist, im Instanzenraum X das *Zentrum* einer Menge von Instanzen zu berechnen, also denjenigen Punkt, der die Summe der Abstände zu allen Punkten dieser Menge minimiert (das Zentrum muss dabei nicht selbst in der Menge enthalten sein). Am einfachsten ist dies, wenn $X = \mathbb{R}^n$, also für rein numerische Instanzenräume. Das Zentrum einer Menge $C \subseteq X$ ist dann einfach das Mittel:

$$z(C) := \frac{1}{|C|} \sum_{\mathbf{c} \in C} \mathbf{c},$$

Der k-Means-Algorithmus arbeitet iterativ wie folgt:

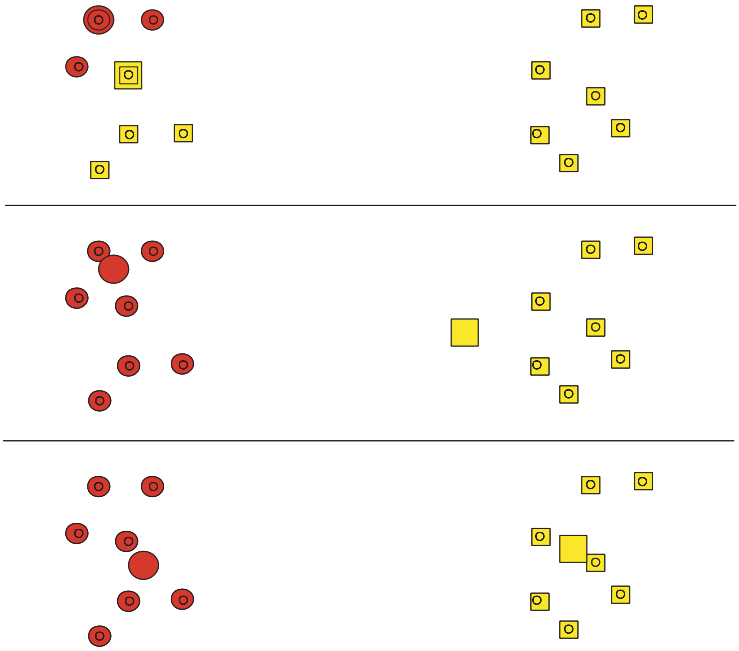

Abbildung 1.18: k-Means Beispiel

proc K-MEANS(S, k)
  $Z = \{z_1, \ldots, z_k\} :=$ k zufällig gewählte Punkte aus S
  while Qualität wird besser
    $C_i := \{s \in S | i = argmin_{i=1,\ldots,k} dist(s, z_i)\}$ für $i = 1, \ldots, k$
    $z_i := z(C_i)$ für $i = 1, \ldots, k$
    end
  return$\{C_1, \ldots, C_k\}$

Es werden also iterativ immer wieder die Zentren der gerade aktuellen Cluster bestimmt, um dann die Cluster auf der Basis dieser Zentren neu zu bilden. Das Verfahren konvergiert sehr schnell, wie auch in Abbildung 1.18 zu sehen ist (die jeweils aktuellen Clusterzentren sind durch den grossen Kreis bzw. das grosse Viereck dargestellt).

Die relevante Qualitätsfunktion beruht hier auf dem durchschnittlichen Abstand zum Clusterzentrum, das Verfahren kann jedoch auch in lokalen Maxima dieser Qualitätsfunktion konvergieren. Da dies von der Menge der anfänglich gewählten Zentren abhängt, sind verschiedene Vorschläge zu deren besserer Initialisierung gemacht worden, bzw. man führt den Algorithmus mehrmals mit unterschiedlich gewählten Anfangszentren bis zur Konvergenz durch. Das Verfahren ist nicht in der Lage, Ausreißer in den Daten zu ignorieren, so dass diese auch als einzelne Punkte dann einen der k Cluster bilden. Eine entsprechende Datenvorverarbeitung zur Entfernung der Ausreißer ist also notwendig. Übliche Implementierungen des k-Means-Algorithmus führen zusätzlich durch wiederholte Läufe noch eine Suche nach dem günstigsten k innerhalb eines vom Benutzer vorgegebenen Bereiches aus.

Es sei schließlich noch angemerkt, dass das k-Means-Verfahren als Instanz eines allgemeinen, aus der Statistik bekannten iterativen Verfahrens zur Schätzung versteckter Parameter betrachtet werden kann, dem sogenannten *EM-Algorithmus*. Im allgemeinen Fall

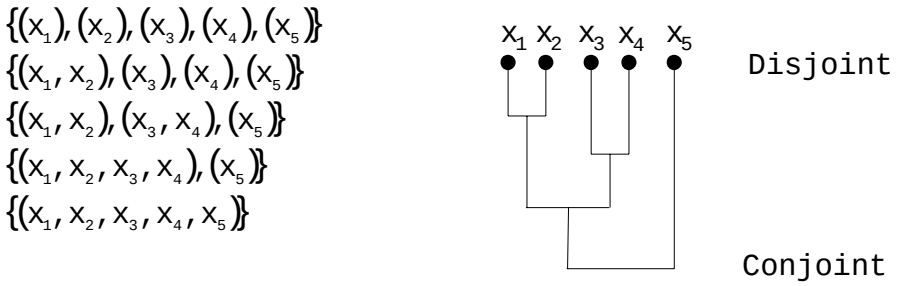

Abbildung 1.19: Hierarchisches Clustering und Dendrogramm

nimmt man dabei an, dass die Datenpunkte in S aus einer Mischung von k verschiedenen Verteilungen erzeugt worden sind, und dass die Parameter dieser Verteilungen und als versteckte Parameter die Clusterzuordnungen zu schätzen sind. Eine solche allgemeinere Modellierung wird beispielsweise vom populären Clusteringverfahren AutoClass [Cheeseman und Stutz, 1996] benutzt.

1.12.2 Hierarchische Clustering-Verfahren

Eine zweite große Gruppe von Clustering-Verfahren sind die *hierarchischen Clusteringverfahren* [Jain und Dubes, 1988]. Bei diesen Verfahren wird nicht nur ein einzelnes Clustering C produziert, sondern eine Serie ineinander geschachtelter Clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_m$. Die beiden Endpunkte dieser Liste geschachtelter Clusterings sind dabei zum einen das aus lauter einelementigen Teilmengen bestehende Clustering, und zum anderen das nur aus einem einzigen Cluster mit allen Elementen aus S bestehende Clustering. Hierarchische Clusterungen werden üblicherweise in sogenannten *Dendrogrammen* dargestellt, siehe Abbildung 1.19.

Eine populäre Verfahrensklasse zur Erzeugung solcher Clusterings ist das *agglomerative Clustern*, das ausgehend von den einelementigen Clustern die Hierarchie aufsteigend aufbaut. Dies geschieht einfach dadurch, dass in jedem Schritt des Verfahrens die beiden einander ähnlichsten Cluster zu einem neuen Cluster zusammengefasst werden, wie das im Dendrogramm entsprechend durch eine waagerechte Linie eingezeichnet ist. Die Ähnlichkeit zweier einelementiger Cluster ist dabei einfach die Ähnlichkeit ihrer beiden jeweils einzigen Elemente. Fasst man zwei Cluster zusammen, kann die Ähnlichkeit des neuen Clusters zu allen bisherigen Clustern immer aus den Ähnlichkeiten der beiden bisherigen Cluster zu allen anderen Clustern berechnet werden. Je nachdem, welche Berechnungsvorschrift man dabei wählt, erhält man unterschiedliche Ergebnisse.

$$dist(C_1 \cup C_2, C_3) := f(dist(C_1, C_3), dist(C_2, C_3))$$

Für $f = min$ ergibt sich das sogenannte *single link* clustering, für $f = max$ das *complete-link* clustering, und für $f = average$ das *average-link* clustering.

Hierarchische Clusteringvefahren haben den Vorteil, dass der Nutzer aus dem Ergebnis des Verfahrens jederzeit ein nicht-hierarchisches Clustering mit der gewünschten Anzahl von Clustern ableiten kann. Durch etwas komplexere Wahl von f ("Ward's function") kann darüber hinaus sichergestellt werden, dass bei jedem einzelnen Aggregationsschritt

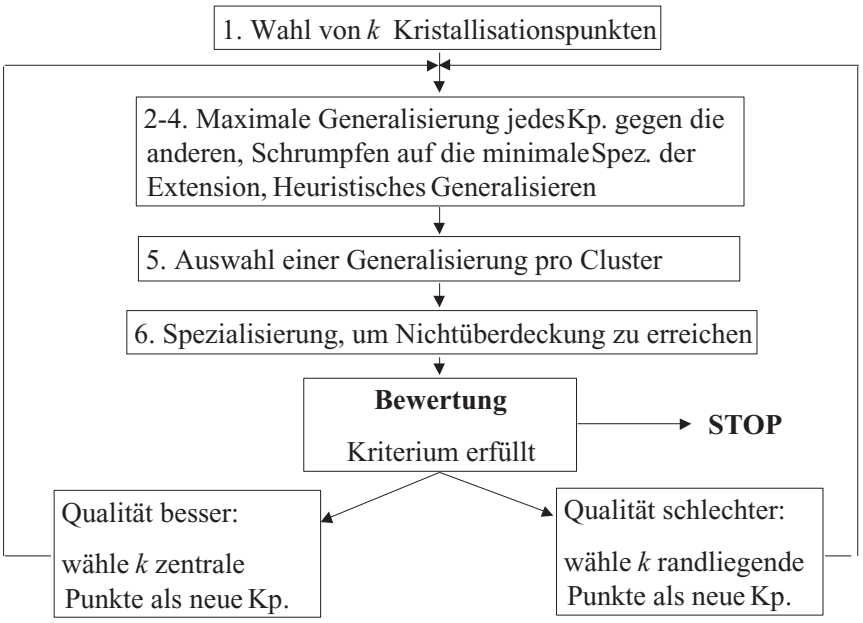

Abbildung 1.20: Der Stern-Algorithmus von CLUSTER

die Summe der (quadrierten) Abstände minimiert wird (dies gilt allerdings nicht für das Gesamtergebnis) [Jain und Dubes, 1988].

1.12.3 Begriffliche Clusteranalyse

Im Prinzip kann aus jedem Cluster-Analyseverfahren ein begriffliches Verfahren gemacht werden, indem man durch ein geeignetes Begriffslernverfahren eine generalisierte Beschreibung jedes Clusters erzeugen lässt, nachdem das eigentliche Clusterverfahren diese Cluster erzeugt hat. Dabei ist jedoch nicht gewährleistet, dass für jeden solchen Cluster auch eine (gute) Beschreibung existiert. Begriffliche Clusterverfahren im engeren Sinne stützen ihre Suche deshalb nicht auf eine Ähnlichkeitsfunktion *dist*, sondern orientieren die Suche an einem spezifizierten Hypothesenraum L_H. Dabei wird wiederum angenommen, dass für jedes Element aus L_H eine Extension, z.B. durch eine Verallgemeinerungsrelation, festgelegt ist.

Einige populäre begriffliche Clusterverfahren bauen die Cluster inkrementell und partionierend auf, indem sie für neu hinzukommende Objekte jeweils entscheiden, ob der Cluster, dem das Objekt zugeordnet wurde, jetzt in zwei Teilcluster aufgespalten werden soll. So entsteht automatisch eine hierarchische Clusterung (siehe z.B. [Gennari *et al.*, 1989]). Ein anderes populäres begriffliches Clusterverfahren, die "Star-Methode" [Michalski und Stepp, 1983], beginnt mit einzelnen *Kristallisationspunkten* (Kp., *seeds*) und nutzt dann die durch die Generalisierungsrelation (bzw. die Teilmengenbeziehungen auf den Extensionen) gegebene Ordnung auf L_H aus, um seine Suche zu organisieren. Wir können den Algorithmus wie in Abbildung 1.20 zusammenfassen[17].

[17] Die Methode heisst "Star"-Methode, weil die in Schritt 2 entstehenden Generalisierungen eines Kristallisationspunktes gegen alle anderen bei einer zweidimensionalen Visualisierung an einen Stern erinnern.

Durch dieses Verfahren wird also *per definitionem* garantiert, dass jeder gefundene Cluster eine Beschreibung aus L_H besitzt, die eine Verallgemeinerung aller Punkte des Clusters ist. Im Verfahren werden an verschiedenen Stellen Heuristiken verwendet, auf die hier nicht näher eingegangen werden soll. Als Qualitätsfunktion wird eine benutzerdefinierte heuristische Funktion verwendet, die sogenannte *lexikographische Evaluierungsfunktion* (LEF), die der Benutzer aus verschiedenen vorgegebenen Kriterien selbst zusammenstellen kann.

1.13 Verstärkungslernen

Mit Verstärkungslernen (engl. Reinforcement Learning) bezeichnet man das Problem welches sich einem Agenten stellt, wenn er durch "Ausprobieren" in einer dynamischen Umgebung optimales Verhalten lernen soll. Beispiele finden sich im Bereich der Robotik, der Steuerung von industriellen Anlagen oder auch bei Spielen. Für das Spiel Backgammon ist TD-Gammon [Tesauro, 1995] das erste Computerprogramm, das auf Weltklasse-Niveau spielt. TD-Gammon hat durch Verstärkungslernen auf 1.5 Millionen Trainingsspielen gegen sich selbst zu spielen gelernt. Die Anwendungen von Verstärkungslernen haben gemein, dass ein Agent komplexe Abfolgen von Aktionen lernen muss. Beim Backgammon sind die Aktionen (möglichst geschickte) Spielzüge, bei mobilen Robotern Kommandos zur Motorsteuerung.

Die Lernaufgabe des Verstärkungslernen unterscheidet sich ganz wesentlich von der des Funktionslernens aus Beispielen. Beim Verstärkungslernen erhält der Agent keine Beispiele von gutem oder schlechtem Verhalten. Stattdessen darf er experimentieren und wird für erfolgreiches Verhalten belohnt (bzw. für Misserfolge bestraft). Hinter diesem Vorgehen steckt die Hypothese, dass es einfacher ist den Agenten auf diese Weise lernen zu lassen, als das Verhalten von Hand zu programmieren und zu warten.

Formal besteht ein Verstärkungslernproblem aus drei Komponenten:

- Eine Menge von Zuständen $\mathcal{S}$: Zu jedem Zeitpunkt befindet sich der Agent in genau einem der Zustände. Die Zustände sind diskret und wir werden im Folgenden annehmen, dass die Menge der Zustände endlich ist. Beim Backgammon ist $\mathcal{S}$ z. B. die Menge aller möglichen Spielpositionen.
- Eine Menge von Aktionen $\mathcal{A}$: Aktionen bringen den Agenten gemäß einer Zustandsübergangsfunktion $\delta : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}$ von einem Zustand zu einem anderen.
- Eine Belohnungsfunktion $r : \mathcal{S} \times \mathcal{A} \longrightarrow \Re$: Jedesmal, wenn der Agent eine Aktion ausführt, erhält er eine (evtl. negative) Belohnung gemäß dieser Belohnungsfunktion. Die Belohnung ist in der Regel eine reelle Zahl.

Diese Komponenten stellen die Umgebung dar, in welcher der Agent agiert. Eine besonders einfache Beispielumgebung ist in Abbildung 1.21 dargestellt. Jedes Kästchen entspricht einem Zustand. Insgesamt gibt es 19 Zustände. Als Aktionen stehen Bewegungen nach *Osten, Westen, Süden, Norden* zur Auswahl. Die Zustandsübergangsfunktion δ gibt den entsprechenden Nachfolgezustand an. Läuft der Agent gegen eine Wand, so bleibt er im aktuellen Zustand. Die Belohnungsfunktion r ist überall null, ausser der Agent bewegt sich in den mit Z markierten Zustand. Für diese Aktion erhält er eine Belohnung von 100.

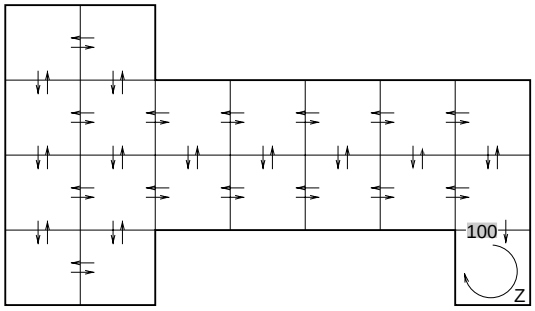

Abbildung 1.21: Eine einfache Beispielumgebung. Jeder Kasten entspricht einem Zustand.

Im Zustand Z gibt es nur eine Aktion - nämlich in ihm zu bleiben. Der Agent erhält keine weiteren Belohnungen. Man nennt einen solchen Zustand *absorbierend*.

Wie interagiert der Agent mit der Umgebung? Von seinem aktuellen Zustand wählt der Agent eine Aktion. Die Auswahl der Aktion geschieht durch die sogenannte Verhaltensregel (engl. policy) $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ des Agenten. Eine gute Verhaltensregel ist das Lernziel des Agenten. Als Rückmeldung auf eine Aktion erhält der Agent seinen neuen Zustand und die Belohnung. Beides hängt lediglich vom aktuellen Zustand und der gewählten Aktion ab, nicht aber von Zuständen oder Aktionen, die weiter in der Vergangenheit liegen. Man bezeichnet dieses Szenario auch als deterministischen Markov'schen Entscheidungsprozess. Dieser Vorgang wiederholt sich solange der Agent lebt. Bis jetzt ist allerdings nicht klar, wie eine gute Verhaltensregel aussehen soll. Intuitiv ist klar, dass der Agent möglichst viele Belohnungen erhalten soll. Aber wie definiert man "möglichst viele Belohnungen" am treffendsten?

1.13.1 Wann handelt ein Agent optimal?

Es gibt mehrere sinnvolle Möglichkeiten, wie man gutes oder optimales Verhalten definieren kann. Hat der Agent eine begrenzte und bekannte Lebenszeit h, so ist es naheliegend, die Summe der Belohnungen zur Bewertung einer Verhaltensregel π heranzuziehen. In diesem Sinne gibt die folgende Funktion $V^{\pi}(s_t)$ die Summe der Belohungen an, welche der Agent erhält, wenn er im Zustand s_t startet und dann der Verhaltensregel π folgt.

$$V^{\pi}_{sum}(s_t) = \sum_{i=0}^{h} r_{t+i}$$

Die optimale Verhaltensregel ist die, welche für alle Startzustände die Funktion $V^{\pi}(s_t)$ maximiert. Ist die Lebenszeit des Agenten nicht beschränkt, ist die durchschnittliche Belohnung ein sinnvolles Kriterium.

$$V^{\pi}_{avg}(s_t) = \lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

Im weiteren Verlauf des Kapitels wird allerdings das folgende Optimalitätskriterium betrachten. Es wird diskontierte kumulative Belohnung (engl. discounted cumulative reward)

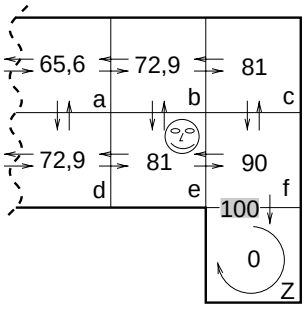

Abbildung 1.22: Die optimalen Werte für $V^*_{dc}(s_t)$ bei $\gamma = 0.9$. Der Agent geht zuerst nach Osten und dann nach Süden.

genannt. Die Idee ist, früh eintretende Belohnungen höher zu bewerten als solche, die sich erst weit in der Zukunft ergeben. Hierzu dient der Parameter $0 \leq \gamma < 1$.

$$V^{\pi}_{dc}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Weiter in der Zukunft liegende Belohnungen werden also exponentiell geringer bewertet. Je kleiner γ ist, desto stärker konzentriert sich die Bewertung auf die sofort eintretende Belohnung. Man kann γ als Zinsrate verstehen, oder auch als Wahrscheinlichkeit, dass der Agent einen weiteren Schritt lebt. Insbesondere sind aber die Algorithmen, welche die diskontierte kumulative Belohnung maximieren, besonders einfach und effizient.

Eine bemerkenswerte Eigenschaft ist, dass es eine Äquivalenz zwischen maximaler Bewertungsfunktion $V^*(s_t)$ und optimaler Verhaltensregel π^* gibt. Es reicht also aus, die Bewertungsfunktion zu maximieren. Die optimale Verhaltensregel lässt sich dann leicht ablesen. Der Agent wählt in einem Zustand s_t die Aktion a, welche die gewichtete Summe aus der sofortigen Belohnung $r(s_t, a)$ und der diskontierten Bewertungsfunktion $V^*_{dc}(\delta(s_t, a))$ des Nachfolgezustandes maximiert.

$$\pi(s_t) \quad = \quad argmax_{a \in \mathcal{A}} \left[r(s_t, a) + \gamma \cdot V^*_{dc}(\delta(s_t, a)) \right]$$

Das Beispiel in Abbildung 1.22 zeigt wie der Agent handelt, wenn er $V^*_{dc}(s_t)$ (hier für $\gamma = 0.9$) kennt. Der Agent befindet sich im Zustand e. Ihm stehen vier Aktionen zur Auswahl. Geht er nach Süden, so läuft er gegen die Wand und bleibt in Zustand e. Die sofortige Belohnung ist 0 und $V^*_{dc}(e) = 81$. Somit ist der Wert der Aktion "Süden" $0 + 0.9 \cdot 81.0 = 72.9$. Ebenso berechnet sich der Wert für die Aktion "Westen" als $0 + 0.9 \cdot 72.9 = 65.6$, für die Aktion "Norden" als $0 + 0.9 \cdot 72.9 = 65.6$ und für die Aktion "Osten" als $0 + 0.9 \cdot 90.0 = 81.0$. Der Agent geht also nach "Osten" und befindet sich nun im Zustand f. Hier stehen wieder vier Aktionen zur Auswahl: "Westen" hat eine Bewertung von $0 + 0.9 \cdot 81.0 = 72.9$, "Norden" $0 + 0.9 \cdot 81.0 = 72.9$, "Osten" $0 + 0.9 \cdot 90.0 = 81.0$ und "Süden" $100.0 + 0.9 \cdot 0.0 = 100.0$. Der Agent geht also nach Süden und bleibt in dem absorbierenden Zustand Z.

Es ist nun klar wie der Agent handelt, wenn er $V^*_{dc}(s_t)$ kennt. Die nächsten zwei Abschnitte behandelt die Aufgabe $V^*_{dc}(s_t)$ zu berechnen, bzw. zu lernen.

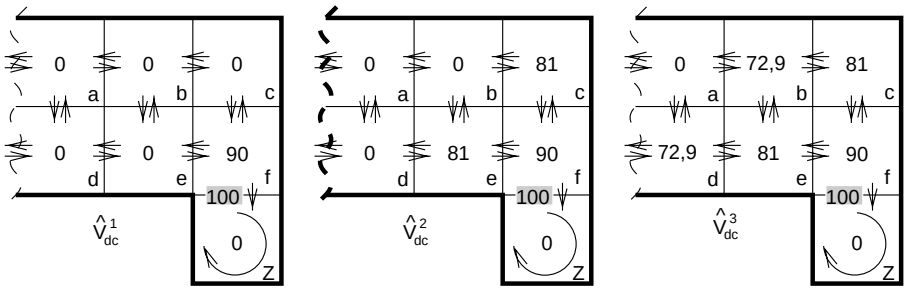

Abbildung 1.23: Die Approximation der Bewertungsfunktion durch den Value-Iteration Algorithmus nach der ersten, zweiten und dritten Iteration für $\gamma = 0.9$.

1.13.2 Dynamische Programmierung

Wie findet man die Verhaltensregel, welche die diskontierte kumulative Belohnung maximiert? Gehen wir zunächst davon aus, dass der Agent die Zustandsübergangsfunktion δ und die Belohnungsfunktion r bereits während des Lernens vollständig kennt. In dieser Situation kann man das Problem mittels dynamischer Programmierung lösen.

Mit Bezug auf das Optimalitätskriterium aus dem vorangegangenen Abschnitt suchen wir die Verhaltensregel π, die für jeden Zustand die Bewertungsfunktion maximiert.

$$V^*_{dc}(s) = max_\pi \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Ein Algorithmus, der diese Verhaltensregel bzw. die maximale Bewertungsfunktion findet, ist der Value-Iteration Algorithmus [Bellman, 1957]. Die Idee des Algorithmus ist die Folgende. Der Algorithmus startet mit einer beliebigen Initialisierung seiner Schätzung $\hat{V}^0_{dc}(s)$ für $V^*_{dc}(s)$. Zum Beispiel ist also $\hat{V}^0_{dc}(s) = 0$ für alle Zustände $s \in \mathcal{S}$ zulässig. Dann iteriert der Algorithmus mehrmals über alle Zustände. Für jeden Zustand s wird die optimale Aktion basierend auf der *aktuellen Schätzung* $\hat{V}^t_{dc}$ berechnet und der Wert $\hat{V}^{t+1}_{dc}(s) = max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \cdot \hat{V}^t_{dc}(\delta(s,a)) \right]$ aktualisiert. Man kann zeigen, das für $t \to \infty$ der Wert $\hat{V}^t_{dc}(s)$ gegen $V^*_{dc}(s)$ konvergiert.

Algorithmus 1 (Value-Iteration)

- *Initialisiere* $\hat{V}^0_{dc}(s) = 0$ *für alle* $s \in \mathcal{S}$, $t = 0$
- *Solange* $\hat{V}^t_{dc}(s)$ *nicht gut genug*
  - *Für alle* $s \in \mathcal{S}$
    - $\hat{V}^{t+1}_{dc}(s) = max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \cdot \hat{V}^t_{dc}(\delta(s,a)) \right]$
  - $t := t + 1$
- *Gib* $\hat{V}^t_{dc}$ *aus*

Abbildung 1.23 zeigt den Fortschritt des Algorithmus in einem Ausschnitt der Beispielumgebung nach dem ersten, zweiten und dritten Durchlauf durch die Schleife.

Es sei an die relativ starken Annahmen erinnert, die der Value-Iteration Algorithmus benötigt - nämlich dass die Zustandsübergangsfunktion δ und die Belohnungsfunktion r bekannt sind. Dies ist bei den meisten Agenten nicht gegeben.

1.13.3 Q-Learning - Lernen in unbekannter Umgebung

In realen Anwendungen sind gerade die Zustandsübergangsfunktion δ und die Belohnungsfunktion r die Teile der Umwelt, die der Agent erst durch Exploration lernen muss. Ein mobiler Roboter, den man in ein unbekanntes Gebäude setzt, weiss nicht in welchem Zustand er sich befindet, wenn er 5 Meter vorwärts durch eine Tür und dann 3 Meter nach rechts geht. Dort könnte eine Wand im Weg sein, oder er könnte eine Treppe hinuntergefallen sein. Desweiteren soll der Roboter erst herausfinden, wofür er Belohnungen erhält. Will ein Agent in unbekannter Umgebung lernen, muss er also Experimente durchführen.

Das erste Problem, welches gelöst werden muss, ist das der Aktionsauswahl. Den Ausdruck $argmax_{a \in \mathcal{A}}[r(s_t, a) + \gamma \cdot V^*_{dc}(\delta(s_t, a))]$ kann man ohne Kenntnis von r und δ nicht auswerten. Man kann dies aber leicht umgehen, indem man eine Bewertungsfunktion $Q(s, a)$ direkt für Aktionen lernt. $Q(s, a)$ ist definiert als $r(s, a) + \gamma \cdot V^*_{dc}(\delta(s, a))$. Kennt man $Q(s, a)$ für alle Aktionen im aktuellen Zustand, so kann man wie gehabt die optimale Aktion leicht bestimmen.

Ein Lernalgorithmus, der Q durch Exploration lernt, ist der sogenannte Q-Learning Algorithmus [Watkins und Dayan, 1992]. Er funktioniert nach einem ähnlichen Prinzip wie der Value-Iteration Algorithmus, benutzt aber nur Informationen, an die der Agent durch Ausprobieren von Aktionen selber gelangen kann. Immer wenn der Agent eine Aktion a ausprobiert, nimmt er seinen neuen Zustand s' und die erhaltene Belohnung r wahr. Die geschätzte Bewertung $\hat{Q}(s, a)$ des Ausgangszustandes verändert er dann nach der Regel

$$\hat{Q}(s, a) := r + \gamma \cdot max_{a' \in \mathcal{A}} \hat{Q}(s', a')$$

Wieder kann man zeigen, dass der Algorithmus die optimale Bewertungsfunktion Q lernt, wenn jede Aktion in jedem Zustand nur häufig genug durchlaufen wird.

Algorithmus 2 (Q-Lernen)

- *Initialisiere* $\hat{Q}(s, a) = 0$ *für alle* $s \in \mathcal{S}$ *und* $a \in \mathcal{A}$, s *ist Startzustand*
- *Wiederhole solange der Agent lebt*
  - *Wähle eine Aktion* a *und führe sie aus*
  - *Erhalte Belohnung* r *und neuen Zustand* s'
  - $\hat{Q}(s, a) := r + \gamma \cdot max_{a' \in \mathcal{A}} \hat{Q}(s', a')$
  - $s := s'$

Die Aktionsauswahl ist im Algorithmus nicht näher spezifiziert solange sichergestellt ist, dass jede Aktion häufig genug ausprobiert wird. Hier ergibt sich das sogenannte "Exploration vs. Exploitation" Dilemma. Zu Beginn des Algorithmus wird der Agent Aktionen zufällig auswählen, da alle die gleiche Bewertung haben. Aber nach einiger Zeit hat der Agent ein passables $\hat{Q}$ gelernt. Sollte der Agent weiterhin beliebige Aktionen ausprobieren und so vielleicht ein besseres $\hat{Q}$ finden (Exploration), oder sollte der Agent immer die Aktion mit dem größten $\hat{Q}(s, a)$ wählen, um die Belohnung zu maximieren (Exploitation)? Oft wird eine Mischung zwischen beidem gewählt. Mit über die Zeit steigender Wahrscheinlichkeit wählt der Agent die Aktion mit dem größten $Q(s, a)$, aber mit einer gewissen Wahrscheinlichkeit auch eine der anderen Aktionen. Eine optimale Lösung dieses Problems ist allerdings schwierig.

1.13.4 Erweiterungen

Aus Platzgründen wurde hier nur die einfachste Form des Verstärkungslernens behandelt. Einige Erweiterungen sollen aber trotzdem kurz erwähnt werden:

Nicht-Deterministische Umgebungen Aktionen wurden als deterministisch angenommen d. h. sie führten immer zum gleichen Nachfolgezustand und zur gleichen Belohnung. Oft ist dies aber nicht der Fall. Z. B. wird ein mobiler Roboter auf den Befehl "gehe einen Meter vorwärts" oft sich nur 80cm weit bewegen, da der Boden an der Stelle uneben ist. Es sind also mehrere Nachfolgezustände möglich, die jeweils mit einer gewissen Wahrscheinlichkeit eintreten. Gleiches kann auch für die Belohnungen gelten. Der Q-Learning Algorithmus kann aber auch diesen Fall handhaben, wenn man die rekursive Formel durch $\hat{Q}(s,a) := (1-\alpha)\hat{Q}(s,a) + \alpha[r + \gamma \cdot max_{a' \in \mathcal{A}} \hat{Q}(s',a')]$ ersetzt, wobei $0 < \alpha \leq 1$ eine Lernrate ist, die langsam gesenkt wird.

Verborgener Zustand Oft kann der Agent seinen Zustand nicht eindeutig bestimmen. Für einen Roboter mögen zwei Räume gleich "aussehen". Die Handhabung dieses Problems ist allerdings sehr schwierig (siehe z. B. [Kaelbling *et al.*, 1996]).

Modellbasiertes Verstärkungslernen Eine Alternative zum Q-Learning ist es, zuerst δ und r zu lernen und dann Algorithmen wie Value-Iteration anzuwenden. Auch Kombinationen beider Methoden werden untersucht.

Funktionslernen Für jeden Zustand eine separate Bewertung zur lernen und speichern ist für große Zustandsmengen zu ineffizient. Indem man Zustände durch Merkmale beschreibt, versucht man die Bewertungsfunktion mit Methoden aus Kapitel 1.2 zu lernen. Gelingt es gut zu generalisieren, kann der Agent auch in Situationen gut handeln, die er vorher noch nicht untersucht hat.

Literatur Als weiterführende Literatur sind [Sutton und Barto, 1998] und in kompakterer Form [Mitchell, 1997] und [Kaelbling *et al.*, 1996] zu empfehlen.

Literaturverzeichnis

[Agrawal *et al.*, 1996] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen und A. Inkeri Verkamo. Fast Discovery of Association Rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth und Ramasamy Uthurusamy (Hg.), *Advances in Knowledge Discovery and Data Mining*, Kapitel 12, Seite 307 – 328. AAAI/MIT Press, Cambridge, USA, 1996.

[Aha *et al.*, 1991] David Aha, Dennis Kibler und Marc Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6:37 – 66, 1991.

[Alchourron *et al.*, 1985] Alchourron, Grädenfors und Makinson. On the logic of the theory change - partial meet contraction and revision functions. *Journal o Symbolic Logic*, 50:510–530, 1985.

[Angluin und Smith, 1983] Dana Angluin und Carl Smith. Inductive Inference: Theory and Methods. *Computing Surveys*, 15:237–269, 1983.

[Anthony und Biggs, 1992] Martin Anthony und Norman Biggs. *Computational Learning Theory*. Univ. Press, Cambridge, Mass. ua., 1992.

[Bacher, 1996] Johann Bacher. *Clusteranalyse*. Oldenbourg, München, 1996.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Boser *et al.*, 1992] Bernhard E. Boser, Isabelle M. Guyon und Vladimir N. Vapnik. A Traininig Algorithm for Optimal Margin Classifiers. In D. Haussler (Hg.), *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, Seite 144–152, 1992.

[Breiman *et al.*, 1984] L. Breiman, J.H. Friedman, R.A. Olshen und C.J. Stone. *Classification and regression trees*. Belmont, Wadsworth, 1984.

[Breiman, 1996] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123 – 140, 1996.

[Buntine, 1988] W. Buntine. *Decision Tree Induction Systems: A Bayesian Analysis*, Band 3, Kapitel II, Seite 109–128. North-Holland, Amsterdam, 1988.

[Burges, 1998] C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Cheeseman und Stutz, 1996] P. Cheeseman und J. Stutz. Bayesian Classification (AutoClass): Theory and Results. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic

Smyth und Ramasamy Uthurusamy (Hg.), *Advances in Knowledge Discovery and Data Mining*, Kapitel 6, Seite 153 – 180. AAAI/MIT Press, Cambridge, USA, 1996.

[Christianini und Shawe-Taylor, 2000] N. Christianini und J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[Cohen und Hirsh, 1992] William W. Cohen und Haym Hirsh. Learnability of Description Logics. *Procs. 5th Annual Conference on Computational Learning Theory*, Seite 116–127, 1992.

[Cortes und Vapnik, 1995] Corinna Cortes und Vladimir N. Vapnik. Support–Vector Networks. *Machine Learning Journal*, 20:273–297, 1995.

[Cover und Hart, 1967] T. M. Cover und P.E. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13:21 – 27, 1967.

[Diehl und Arbinger, 1990] Joerg M. Diehl und Roland Arbinger. *Einführung in die Inferenzstatistik*. D. Klotz Verlag, Eschborn, 1990.

[Džeroski *et al.*, 1992] Sašo Džeroski, Stephen Muggleton und Stuart Russell. PAC-Learnability of Determinate Logic Programs. *Procs. of 5th COLT*, Seite 128–135, 1992.

[Emde und Wettschereck, 1996] Werner Emde und Dietrich Wettschereck. Rational instance based learning. In Lorenza Saitta (Hg.), *Proceedings 13th International Conference on Machine Learning*, Seite 122–130. Morgan Kaufmann, 1996.

[Fayyad *et al.*, 1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro und Padhraic Smyth. From Data Mining to Knowledge Discovery: An Overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth und Ramasamy Uthurusamy (Hg.), *Advances in Knowledge Discovery and Data Mining*, Kapitel 1, Seite 1 – 34. AAAI/MIT Press, Cambridge, USA, 1996.

[Frazier und Pitt, 1994] Michael Frazier und Leonard Pitt. Classic Learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, Seite 23–34, 1994.

[Gärdenfors, 1988] Peter Gärdenfors. *Knowledge in Flux — Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA, 1988.

[Gebhardt, 1991] Friedrich Gebhardt. Choosing Among Competing Generalizations. *Knowledge Acquisition*, 3:361 – 380, 1991.

[Gennari *et al.*, 1989] John H. Gennari, Pat Langley und Doug Fisher. Models of Incremental Concept Formation. *Artificial Intelligence*, 40:11 – 61, 1989.

[Goldberg und Holland, 1988] D.E. Goldberg und J.H. Holland. Genetic Algorithms and Machine Learning. *Machine Learning*, 3(2/3):95–100, 1988.

[Haussler, 1988] David Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221, 1988.

[Hoffmann, 1991] Achim Hoffmann. Die Theorie des Lernbaren - ein Überblick. *KI*, 1:7–11, 1991.

[Horvath *et al.*, 2000] Tamas Horvath, Stefan Wrobel und Uta Bohnebeck. Rational instance-based learning with lists and terms. *Machine Learning Journal*, 2000. to appear.

[Jain und Dubes, 1988] A.K. Jain und R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, New York, 1988.

[Joachims, 1998] T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *Proceedings of the European Conference on Machine Learning*, Berlin, Seite 137 – 142. Springer, 1998.

[Joachims, 1999] T. Joachims. Making large-Scale SVM Learning Practical. In B. Schölkopf, C. Burges und A. Smola (Hg.), *Advances in Kernel Methods - Support Vector Learning*, Kapitel 11. MIT Press, Cambridge, MA, 1999.

[Jr. und Frisch, 1992] C. D. Page Jr. und A. M. Frisch. Generalization and Learnability : A Study of Constrained Atoms. In Stephen Muggleton (Hg.), *Inductive Logic Programming.*, Kapitel 2, Seite 29–62. Academic Press, The A.P.I.C. Series 38, London [u.a.], 1992.

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L. Littman und Andrew W. Moore. Reinforcement Learning: A Survey. *Artificial Intelligence Research*, 4:237–285, may 1996.

[Kearns *et al.*, 1997] M. Kearns, Y. Mansour, A. Ng und D. Ron. An experimental and theoretical comparison of model selection methods. *Machine Learning*, 27:7 – 50, 1997.

[Kearns und Vazirani, 1994] Michael Kearns und Umesh Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[Kearns, 1990] Michael Kearns. *The Computational Complexity of Machine Learning*. The MIT Press, ACM Distinguished Dissertation, 1990.

[Kietz und Lübbe, 1994] Jörg-Uwe Kietz und Marcus Lübbe. An Efficient Subsumption Algorithm for Inductive Logic Programming. In W. Cohen und H. Hirsh (Hg.), *Proceedings of the 11th International Conference on Machine Learning IML–94*, San Francisco, CA. Morgan Kaufmann, 1994.

[Kietz und Morik, 1994] Jörg-Uwe Kietz und Katharina Morik. A Polynomial Approach to the Constructive Induction of Structural Knowledge. *Machine Learning Journal*, 14(2):193 – 217, feb 1994.

[Kietz und Wrobel, 1992] J.-U. Kietz und S. Wrobel. Controlling the Complexity of Learning in Logic through Syntactic and Task–Oriented Models. In Stephen Muggleton (Hg.), *Inductive Logic Programming.*, Kapitel 16, Seite 335–360. Academic Press, The A.P.I.C. Series 38, London [u.a.], 1992.

[Kietz, 1993] Jörg-Uwe Kietz. Some Lower Bounds for the Computational Complexity of Inductive Logic Programming. In Pavel Brazdil (Hg.), *Proceedings of the Sixth European Conference on Machine Learning ECML-93*, Berlin, Heidelberg, New York, Seite 115–123. Springer, Lecture Notes in Artificial Intelligence, 1993. Also available as Arbeitspapiere der GMD No. 718, 1992.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C.D. Gelatt und M.P. Vercchi. Optimization by Simulated Annealing. *Science*, 220:671 – 680, 1983.

[Kirsten *et al.*, 1998] Mathias Kirsten, Stefan Wrobel, Friedrich-Wilhelm Dahmen und Hans-Christoph Dahmen. Einsatz von Data-Mining-Techniken zur Analyse Ökologischer Standort- und Pflanzendaten. *Künstliche Intelligenz Journal*, 12(2):39–42, 1998.

[Klösgen, 1996] Willi Klösgen. Explora: A Multipattern and Multistrategy Discovery Assistant. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth und Ramasamy Uthurusamy (Hg.), *Advances in Knowledge Discovery and Data Mining*, Kapitel 10, Seite 249 – 271. AAAI/MIT Press, Cambridge, USA, 1996.

[Klösgen, 2000a] Willi Klösgen. Change analysis. In Willi Klösgen und Jan Zytkow (Hg.), *Handbook of Knowledge Discovery and Data Mining*. Oxford University Press, London, 2000. to appear.

[Klösgen, 2000b] Willi Klösgen. Deviation Analysis. In Willi Klösgen und Jan Zytkow (Hg.), *Handbook of Knowledge Discovery and Data Mining*. Oxford University Press, London, 2000. to appear.

[Klösgen, 2000c] Willi Klösgen. Subgroup patterns. In Willi Klösgen und Jan Zytkow (Hg.), *Handbook of Knowledge Discovery and Data Mining*. Oxford University Press, London, 2000. to appear.

[Lange und Zeugmann, 1992] Steffen Lange und Thomas Zeugmann. Types of Monotonic Language Learning and Their Characterization. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT '92)*, New York, Seite 377–390. ACM Press, 1992.

[Lavrač und Džeroski, 1994] Nada Lavrač und Sašo Džeroski. *Inductive Logic Programming — Techniques and Applications*. Ellis Horwood, Hertfortshire, 1994.

[Lunts und Brailovskiy, 1967] A. Lunts und V. Brailovskiy. Evaluation of Attributes Obtained in Statistical Decision Rules. *Engineering Cybernetics*, 3:98–109, 1967.

[MacQueen, 1967] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, 1967.

[Michalski und Stepp, 1983] Ryszard S. Michalski und Robert E. Stepp. Learning from Observation: Conceptual Clustering. In R.S. Michalski, J.G. Carbonell und T.M. Mitchell (Hg.), *Machine Learning — An Artificial Intelligence Approach, Vol. I*, Band I, Seite 331 – 363. Tioga, Palo Alto, CA, 1983.

[Michalski, 1986] Ryszard Michalski. Understanding the Nature of Learning. In Michalski, Carbonell und Mitchell (Hg.), *Machine Learning - An Artificial Intelligence Approach*. Morgan Kaufmann, Los Altos, California, 1986.

[Mingers, 1989] John Mingers. An Empirical Comparison of Pruning Methods for Decision Tree Induction. *Machine Learning*, 4:227 – 243, 1989.

[Mitchell, 1982] Tom M. Mitchell. Generalization as Search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

[Muggleton und Bain, 1992] S. Muggleton und M. Bain. Non-Monotonic Learning. In Stephen Muggleton (Hg.), *Inductive Logic Programming*, Kapitel 13, Seite 281–298. Academic Press, London, 1992.

[Muggleton und Feng, 1992] S. Muggleton und C. Feng. Efficient Induction of Logic Programs. In Stephen Muggleton (Hg.), *Inductive Logic Programming.*, Kapitel 13, Seite 281–298. Academic Press, The A.P.I.C. Series 38, London [u.a.], 1992.

[Muggleton, 1992] Stephen Muggleton (Hg.). *Inductive Logic Programming.* Academic Press, 1992.

[Murthy *et al.*, 1994] Sreerama K. Murthy, Simon Kasif und Steven Salzberg. A System for Induction of Oblique Decision Trees. *Journal of Artificial Intelligence Research*, 2:1 – 32, 1994.

[Murthy, 1998] Sreerama K. Murthy. Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey. *Data Mining and Knowledge Discovery*, 2(4):345 – 389, 1998.

[Natarajan, 1991] Balas K. Natarajan. *Machine Learning. A Theoretical Approach.* Morgan Kaufmann, San Mateo, CA, 1991.

[Nebel, 1989] Bernhard Nebel. A Knowledge Level Analysis of Belief Revision. In *Proc. of the 1st Int. Conf. on Principles of Knowledge Representation and Reasoning*, Seite 301–311, 1989.

[Nienhuys-Cheng *et al.*, 1993] S.-H. Nienhuys-Cheng, P.R.J. van der Laag und L.W.N. van der Torre. Constructing refinement operators by decomposing logical implication. In *Third Congres of the Italian Association for Artificial Intelligence*, Seite 178–189. Springer Verlag, 1993.

[Nienhuys-Cheng und Wolf, 1997] Shan-Hwei Nienhuys-Cheng und Ronald de Wolf. *Foundations of Inductive Logic Programming.* Springer Verlag, LNAI Tutorial 1228, Berlin, New York, 1997.

[Platt, 1999] J. Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In B. Schölkopf, C. Burges und A. Smola (Hg.), *Advances in Kernel Methods - Support Vector Learning*, Kapitel 12. MIT-Press, 1999.

[Plotkin, 1970] Gordon D. Plotkin. A note on inductive generalization. In B. Meltzer und D. Michie (Hg.), *Machine Intelligence*, Kapitel 8, Seite 153–163. American Elsevier, 1970.

[Plotkin, 1971] Gordon D. Plotkin. A further note on inductive generalization. In B. Meltzer und D. Michie (Hg.), *Machine Intelligence*, Kapitel 8, Seite 101–124. American Elsevier, 1971.

[Quinlan, 1990] J.R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5(3):239–266, 1990.

[Quinlan, 1993] J. Ross Quinlan. *C4.5 — Programs for Machine Learning.* Morgan Kaufman, San Mateo, CA, 1993. Accompanying software available.

[Rencher, 1998] Alvin C. Rencher. *Multivariate Statistical Inference and Applications.* John Wiley, New York, 1998.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. e. Hinton und R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart und J. E. McClelland

(Hg.), *Parallel distriguted systems: Explorations in the microstructure of cognition (Vol. I & II)*, Seite 318 – 362 (Vol. I). MIT Press, Cambridge, MA, 1986.

[Salzberg, 1991] S. Salzberg. A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6:277–309, 1991.

[Schapire, 1999] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[Scheffer und Joachims, 1999] Tobias Scheffer und Thorsten Joachims. Expected Error Analysis for Model Selection. In Ivan Bratko und Sašo Džeroski (Hg.), *Proc. 16th Int. Conf. on Machine Learning*, San Mateo, CA. Morgan Kaufman, 1999.

[Shafer *et al.*, 1996] John C. Shafer, Rakesh Agrawal und Manish Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan und N. L. Sarda (Hg.), *Proc. 22th Int. Conf. on Very Large Data Bases (VLDB96)*, San Mateo, CA, Seite 544 – 555. Morgan Kaufman, 1996.

[Shannon und Weaver, 1969] C.E. Shannon und W. Weaver. *The Mathematical Theory of Communication*, Seite 20–21. Chapman and Hall, 4 Auflage, September 1969.

[Shapiro, 1981] E.Y. Shapiro. An Algorithm that Infers Theories from Facts. In *Proc of the seventh IJCAI-81*, Seite 446–451, 1981.

[Shapiro, 1983] Ehud Y. Shapiro. *Algorithmic Program Debugging.* The MIT Press, ACM Distinguished Doctoral Dissertations, Cambridge, Mass., 1983.

[Simon, 1983] Herbert A. Simon. Why Should Machines Learn? In R. S. Michalski, J. G. Carbonell und T. M. Mitchell (Hg.), *Machine Learning — An Artificial Intelligence Approach*, Band 1, Kapitel 2, Seite 25–39. Morgan Kaufmann, Palo Alto, CA, 1983.

[Srikant und Agrawal, 1995] Ramakrishnan Srikant und Rakesh Agrawal. Mining Generalized Association Rules. In Umeshwar Dayal, Peter M. D. Gray und Shojiro Nishio (Hg.), *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, Seite 407–419. Morgan Kaufmann, 1995.

[Srikant und Agrawal, 1996] Ramakrishnan Srikant und Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Peter M. G. Apers, Mokrane Bouzeghoub und Georges Gardarin (Hg.), *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings*, Band 1057 Lecture Notes in Computer Science, Seite 3 – 17. Springer Verlag, 1996.

[Steinhausen und Langer, 1977] Detlev Steinhausen und Klaus Langer. *Clusteranalyse.* Walter DeGruyter, Berlin, New York, 1977.

[Sutton und Barto, 1998] R. Sutton und G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.

[Tesauro, 1995] G. Tesauro. Temporal Difference Learning and TD-GAMMON. *Communications of the ACM*, 38(3):58–68, 1995.

[Valiant, 1984] Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[van der Laag, 1995] Patrick van der Laag. *An Analysis of Refinement Operators in Inductive Logic Programming.* Tinbergen Institute Research Series, Rotterdam, 1995.

[Vapnik, 1998] V. Vapnik. *Statistical Learning Theory.* Wiley, Chichester, GB, 1998.

[Watkins und Dayan, 1992] C. Watkins und P. Dayan. Technical Note: Q-Learning. *Machine Learning*, 8(3):279–292, May 1992.

[Wettschereck *et al.*, 1997] D. Wettschereck, D.W. Aha und T. Mohri. A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial Intelligence Review*, 11:273 – 314, 1997.

[Wettschereck und Dietterich, 1995] D. Wettschereck und T.G. Dietterich. An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, 19:5–27, 1995.

[Wrobel, 1993] Stefan Wrobel. On the Proper Definition of Minimality in Specialization and Theory Revision. In Pavel Brazdil (Hg.), *Machine Learning - ECML-93*, Seite 65–82. Springer, Berlin, Heidelberg, New York, 1993.

[Wrobel, 1994] Stefan Wrobel. Wartungsunterstützung im integrierten System MOBAL. In F. Lehner (Hg.), *Die Wartung von Wissensbasierten Systemen,*, Seite 180–200. Egelsbach:Haensel-Hohenhausen, 1994. ISBN 3-89349-561-4.

[Wrobel, 1997] Stefan Wrobel. An Algorithm for Multi–relational Discovery of Subgroups. In J. Komorowski und J. Zytkow (Hg.), *Principles of Data Minig and Knowledge Discovery: First European Symposium (PKDD 97)*, Berlin, New York, Seite 78–87. Springer, 1997.

[Wrobel, 1998] Stefan Wrobel. Data Mining und Wissensentdeckung in Datenbanken. *Künstliche Intelligenz*, 12(1):6–10, 1998.

Liste der Autoren

Clemens Beckstein	Universität Erlangen-Nürnberg, IMMD 8 – Lehrstuhl für KI
Gerhard Brewka	G. Brewka, Universitaet Leipzig, Institut fuer Informatik, Leipzig
Stephan Busemann	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Saarbrücken
Leonie Dreschler-Fischer	Universität Hamburg, Fachbereich Informatik
Günther Görz	Universität Erlangen-Nürnberg, IMMD 8 – Lehrstuhl für KI
Christopher Habel	Universität Hamburg, Fachbereich Informatik
Barbara Hemforth	Universität Freiburg, Institut für Informatik und Gesellschaft, Lehrstuhl für Kognitionswissenschaft
Hagen Langer	Universität Osnabrück, Fachbereich 7
Kai v. Luck	Fachhochschule Hamburg, Fachbereich Elektrotechnik und Informatik
Hanspeter A. Mallot	Max-Planck-Institut für biologische Kybernetik, Tübingen
Katharina Morik	Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII
Sven Naumann	Universität Trier, LDV/CL
Bernhard Nebel	Universität Ulm, Fakultät für Informatik
Bernd Neumann	Universität Hamburg, Fachbereich Informatik
Bernd Owsnicki-Klewe	Fachhochschule Hamburg, Fachbereich Elektrotechnik und Informatik
Simone Pribbenow	Universität Hamburg, Fachbereich Informatik
Frank Puppe	Universität Würzburg, Lehrstuhl für Informatik VI
Michael M. Richter	Universität Kaiserslautern
Josef Schneeberger	SCHEMA GmbH, Nürnberg
Gerhard Strube	Universität Freiburg, Institut für Informatik und Gesellschaft, Lehrstuhl für Kognitionswissenschaft
Peter Struß	Technische Universität München, Institut für Informatik
Ipke Wachsmuth	Universität Bielefeld, Technische Fakultät

Christoph Walther Technische Hochschule Darmstadt, Fachbereich Informatik

Index