Seminar Business Intelligence (2) – Data Mining & Knowledge Discovery

Thema: Klassifikation und Prädiktion

Ausarbeitung von Philipp Breitbach

AG DBIS
Betreuung: Jernej Kovse

Inhaltsverzeichnis

Kapitel 1: Einleitung

Große Unternehmen verwalten riesige Mengen an Daten über Kunden, Produkte, Verkäufe etc. In diesen Daten ist ein enormes Potential an Wissen versteckt, welches sich aber für den Menschen nicht durch bloßes „Anschauen“ erschließt, da er bei der schieren Masse der Daten den Überblick verliert. *Data-Mining*-Anwendungen versuchen dieses versteckte Wissen automatisch aus den Daten zu extrahieren und es dem Analysten in Form von Regeln zugänglich zu machen. Unternehmen nutzen dieses Wissen zur strategischen Entscheidungsfindung, um sich so Wettbewerbsvorteile gegenüber ihren Konkurrenten zu erarbeiten. Data-Mining-Anwendungen werden insbesondere im wirtschaftlichen Bereich eingesetzt, so zum Beispiel bei Marketing und Verkauf zur Analyse des Konsumverhaltens von Kunden oder bei der Vergabe von Versicherungen und Krediten. Aber auch in anderen Bereichen, wie in der medizinischen Diagnose, kann das Data Mining Fuß fassen.

Im Data Mining werden vielfältige Konzepte angewandt, um Beziehungen, Regeln oder Ähnlichkeiten in den vorhandenen Datenbeständen zu entdecken. Eines dieser Konzepte ist die *Klassifikation*, die Datensätze in verschiedene Klassen einteilt. Ein weiteres vom Data Mining genutztes Konzept ist die *Prädiktion*, die Attributwerte von Datensätzen vorhersagt. Beide Konzepte können zur Vorhersage fehlender Attributwerte benutzt werden: die Klassifikation sagt eine Klasse voraus, zu der der entsprechende Datensatz gehört, während die Prädiktion einen numerischen Attributwert bestimmt. Außerdem kann die Klassifikation zusätzlich zur Erstellung von Regeln benutzt werden, die Aussagen über den Zusammenhang zwischen den Attributwerten eines Satzes und seiner Klasse treffen.

Es existiert eine große Vielfalt von Klassifikationsverfahren wie Entscheidungsbauminduktion, naive Bayes’sche Klassifikation oder Neuronale Netze und von Prädiktionsverfahren wie Lineare Regression, die ihren Ursprung vor allem in Statistik und Künstlicher Intelligenz haben. Prinzipiell sind fast alle vorgestellten Verfahren auch für das Data Mining anwendbar. Jedoch muss das Data Mining aufgrund seiner enorm großen Datenmengen die meisten dieser Verfahren weiterentwickeln.

Diese Ausarbeitung wird sich mit den beiden populärsten Klassifikationsverfahren beschäftigen: die *Entscheidungsbauminduktion* und die *Bayes’sche Klassifikation*. Dazu werden zunächst in Kapitel 2 die allgemeinen Vorgehensweisen der Klassifikation und Prädiktion erläutert. Darauf aufsetzend wird Kapitel 3 ausführlich die Entscheidungsbauminduktion und ihre Weiterentwicklung für die speziellen Anforderungen des Data Mining darstellen, während Kapitel 4 eine Beschreibung der Bayes’schen Klassifikation beinhaltet. Kapitel 5 schließlich beschäftigt sich mit der Prädiktion in Form der linearen und nichtlinearen Regression.

An dieser Stelle sei noch gesagt, dass es natürlich wesentlich mehr hochinteressante Klassifikationsverfahren gibt als die hier dargestellten. Sie können jedoch hier aufgrund des begrenzten Rahmens dieser Ausarbeitung nicht alle eingehend bearbeitet werden, und so ziehe ich es vor zwei dieser Verfahren detailliert zu beschreiben anstatt beispielsweise vier nur oberflächlich. Für eine Beschreibung des Backpropagation-Algorithmus Neuronaler Netze sei allerdings noch auf [HK01] verwiesen

Kapitel 2: Grundlagen, Abgrenzung und Begriffsbildung

In diesem Kapitel wird zunächst eine Abgrenzung der Begriffe Klassifikation und Prädiktion, wie sie in dieser Ausarbeitung verwendet werden, vorgenommen. Als nächstes wird das grundlegende Konzept der Klassifikation vorgestellt. Zudem werden begleitend die wichtigsten Fachbegriffe definiert.

Sowohl bei der Klassifikation als auch bei der Prädiktion werden unbekannte Eigenschaften von Datenobjekten vorhergesagt. Die Unterscheidung besteht nun in den Wertebereichen dieser vorherzusagenden Eigenschaften. Die *Klassifikation* sagt Eigenschaften mit endlichen Wertebereichen voraus, sie teilt also ein Objekt einer Kategorie oder Klasse zu. Ein typisches Beispiel für eine kategorische Eigenschaft ist die Farbe. Die *Prädiktion* hingegen prognostiziert den Wert von solchen Eigenschaften, die unendlich viele Werte annehmen können, wie z.B. die vorrausichtliche Absatzmenge im laufenden Geschäftsjahr. Hier wird vor allem die Klassifikation betrachtet, während die Prädiktion eher kurz diskutiert wird. Deswegen soll im Folgenden das grundlegende Konzept der Klassifikation betrachtet werden:

Der *Klassifikationsprozess* besteht aus zwei Schritten. Im *ersten Schritt* wird ein Modell gebildet, welches vorher festgelegte Klassen von Daten beschreibt. Die Erstellung dieses Modells erfolgt durch die *Analyse* von Datentupeln, die durch Eigenschaften, *Attribute* genannt, beschrieben werden. Die Attribute unterscheiden sich im Umfang ihres Wertebereichs. Attribute mit unendlichem Wertebereich heißen *stetige Attribute*, während Attribute mit endlichem Wertebereich wie oben schon angedeutet *kategorische Attribute* sind.

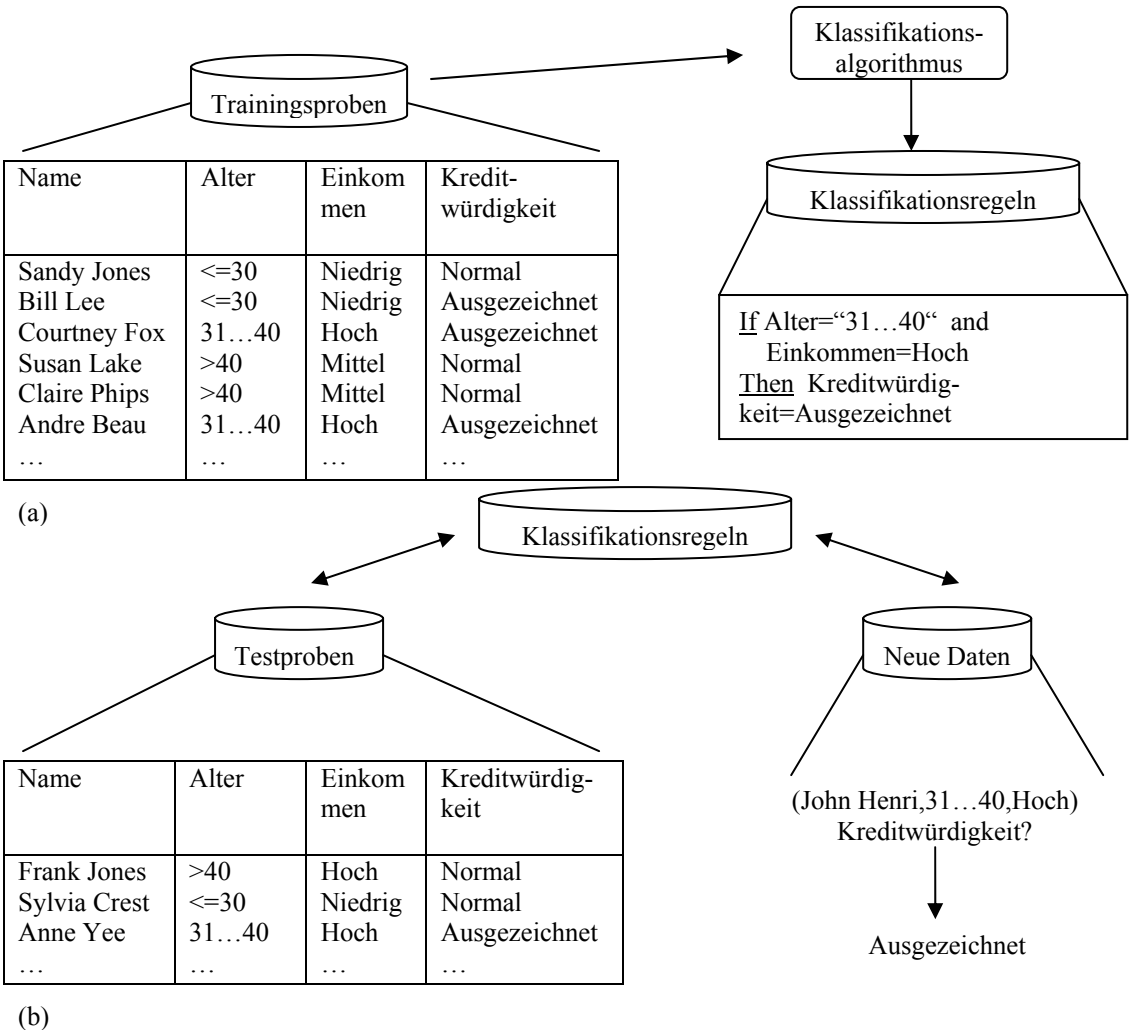
Es wird angenommen, dass jedes Tupel zu einer vordefinierten Klasse gehört. Diese wird durch eines der Attribute bestimmt, das so genannte *klassenbestimmende Attribut*. Das so erlernte Modell wird als *Klassifikator* bezeichnet und kann durch einen Entscheidungsbaum, einen Bayes'scher Klassifikator, ein Neuronales Netz u.a. verwirklicht werden. In dieser Ausarbeitung wird die Verwendung von Entscheidungsbäumen und Bayes'schen Klassifikatoren betrachtet.

Datentupel werden im Zusammenhang mit der Klassifikation häufig auch als *Proben* bezeichnet. Weiterhin bilden diejenigen Proben, die zur Erstellung des Klassifikators analysiert werden, die *Menge der Trainingsdaten*. Die einzelnen Elemente der Menge der Trainingsdaten werden als *Trainingsproben* bezeichnet. Sie werden zufällig aus der Menge aller Proben ausgewählt. Da für alle Trainingsproben bekannt ist, zu welcher Klasse sie gehören, wird dieser Schritt auch als *überwachtes Lernen* bezeichnet. Bei nicht bekannter Klassenzugehörigkeit, oder falls gar die Menge der Klassen unbekannt ist, spricht man von *unüberwachtem Lernen* oder *Clustering*.

Typischerweise wird das aus dem ersten Schritt hervorgehende Modell in Form von Klassifikationsregeln, Entscheidungsbäumen oder mathematischen Formeln dargestellt. In Abbildung 2.1(a) wird die Gewinnung von Klassifikationsregeln aus einer Datenbank mit Informationen über die Kreditwürdigkeit von Kunden, um die Kreditwürdigkeit von neuen Kunden vorherzusagen, dargestellt. Diese Regeln können zur Kategorisierung zukünftiger Proben, aber auch zum Erlangen eines besseren Verständnisses über den Inhalt einer Datenbank verwendet werden.

Im *zweiten Schritt* wird das aus Schritt 1 resultierende Modell zur Klassifikation benutzt. Dabei wird zunächst die Vorhersagegenauigkeit des Modells bzw. des Klassifikators geschätzt. Dazu wird eine Menge von *Testproben* herangezogen, deren klassenbestimmendes Attribut bekannt ist. Die von einer gegebenen Testmenge abgeleitete *Genauigkeit* eines Klassifikators wird berechnet als der Prozentsatz von Proben der Testmenge, die korrekt klassifiziert wurden. Dabei wird für jede Testprobe die bekannte Klasse mit der Klassenvorhersage des Klassifikators verglichen.

Im zweiten Schritt ist zu beachten, dass die Proben der Testmenge zufällig und unabhängig von den Proben der Trainingsmenge ausgewählt werden müssen. Falls die Genauigkeit des Klassifikators mithilfe der Trainingsmenge ermittelt würde, so würde man einen optimistischen Schätzwert für die Genauigkeit erhalten, weil alle Lernverfahren dazu neigen, sich den Trainingsdaten so sehr anzupassen, dass sie Anomalien in der Trainingsmenge, die in der gesamten Menge aller Proben nicht auftreten, widerspiegeln. Deswegen wird eine unabhängige Testmenge verwendet.



Name	Alter	Einkommen	Kreditwürdigkeit
Sandy Jones	<=30	Niedrig	Normal
Bill Lee	<=30	Niedrig	Ausgezeichnet
Courtney Fox	31…40	Hoch	Ausgezeichnet
Susan Lake	>40	Mittel	Normal
Claire Phips	>40	Mittel	Normal
Andre Beau	31…40	Hoch	Ausgezeichnet
…	…	…	…

Name	Alter	Einkommen	Kreditwürdigkeit
Frank Jones	>40	Hoch	Normal
Sylvia Crest	<=30	Niedrig	Normal
Anne Yee	31…40	Hoch	Ausgezeichnet
…	…	…	…

Abbildung 2.1: Der Klassifikationsprozess. (a) *Analyse* der Trainingsproben durch einen Klassifikations-Algorithmus und Repräsentation des resultierenden Klassifikators durch Klassifikationsregeln. (b) *Klassifikation* der Testproben zur Bestimmung der Genauigkeit der Klassifikationsregeln. Bei genügender Genauigkeit Klassifikation von neuen Daten.

Sobald die Genauigkeit des Klassifikators den Ansprüchen genügt, kann er genutzt werden um zukünftige oder unbekannte Proben zu klassifizieren. Der zweite Schritt des Klassifikationsprozesses ist in Abbildung 2.1(b) schematisch abgebildet.

Kapitel 3: Induktion von Entscheidungsbäumen

Um die Induktion von Entscheidungsbäumen zur Klassifikation zu nutzen, soll hier zunächst kurz das allgemeine Konzept eines Entscheidungsbaums erläutert werden:

Entscheidungsbäume werden genutzt, um Objekte anschaulich verschiedenen Klassen zuzuteilen. Jeder innere Knoten repräsentiert einen Test auf ein Merkmal (*Attribut*) eines Objekts, das Testattribut genannt wird, während ein Zweig das Ergebnis eines solchen Tests zeigt. Die Blätter des Entscheidungsbaumes stehen für die resultierenden Klassen. Ein typischer Entscheidungsbaum ist in Abbildung 3.1 dargestellt. Er stellt die Klasseneinteilung für das klassenbestimmende Merkmal *kauft_Computer* dar und sagt dadurch voraus, ob ein Kunde von *AllElectronics* wahrscheinlich einen Computer kauft oder nicht. Innere Knoten sind als Rechtecke, Blätter als Ellipsen gekennzeichnet.

Um ein unbekanntes Objekt zu klassifizieren, müssen nun lediglich seine Attributwerte wie im Entscheidungsbaum beschrieben getestet werden. Dabei wird ein Pfad von der Wurzel bis zu einem Blatt befolgt, welches die Klasseneinteilung des Objekts enthält. Offensichtlich lassen sich Entscheidungs-

bäume leicht zu Klassifikationsregeln konvertieren, indem für jeden Ast die Tests mit ihren zugehörigen Ergebnissen als Bedingungen und das Blatt als Folgerung betrachtet werden.

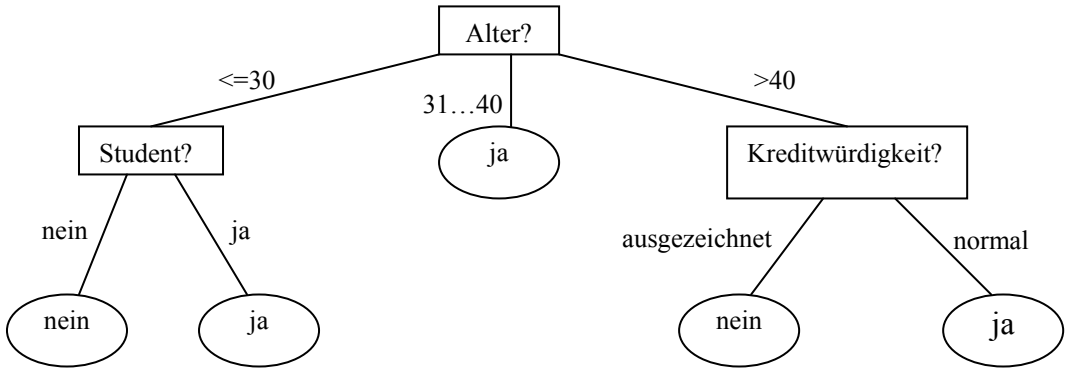


Abbildung 3.1: Ein Entscheidungsbaum für die Einteilung in die beiden Klassen *kauft_Computer* = ja und *kauft_Computer* = nein.

Nun stellt sich die Frage, wie man einen solchen Entscheidungsbaum, der als Klassifikator genügend genaue Ergebnisse liefert, finden kann. Es hat sich eine Methode durchgesetzt, nämlich die Induktion von Entscheidungsbäumen. Induktion meint, dass der Entscheidungsbaum, ausgehend von einer repräsentativen Menge von Trainingsproben, von der Wurzel bis zum Blatt Knoten für Knoten aufgebaut wird.

Abschnitt 3.1 stellt dazu zunächst einen grundlegenden Basisalgorithmus zur Induktion von Entscheidungsbäumen vor. Um einerseits die Qualität in Bezug auf seine Vorhersagegenauigkeit und andererseits die Schnelligkeit des Algorithmus zu verbessern, befasst sich Abschnitt 3.2 mit der Möglichkeit der Baumbeschneidung (*Tree Pruning*). Abschnitt 3.3 schließlich befasst sich mit skalierbaren Algorithmen zur Induktion von Entscheidungsbäumen.

3.1 Der Basisalgorithmus

Der Basisalgorithmus für Entscheidungsbauminduktion ist ein Greedy-Algorithmus, der einen Entscheidungsbaum, wie in Abbildung 3.2 dargestellt, mit einem Top-Down-Ansatz rekursiv aufbaut. Er ist eine Version von ID3[Qui86], einem altbekannten Algorithmus für die Induktion von Entscheidungsbäumen. Die Beschreibung seiner Funktionsweise wird im Anschluss durch ein Beispiel ergänzt, das die Berechnung des Informationsgewinns und die in einer Rekursionsstufe durchgeführten Schritte des Algorithmus näher bringt.

Der Basisalgorithmus kann nur auf kategorische Attribute angewendet werden. Deshalb müssen alle vorhandenen stetigen Attribute kategorisiert werden, d.h. die unendlichen Wertebereiche der stetigen Attribute müssen in endlich viele disjunkte Teilmengen aufgeteilt werden. Im später dargestellten Beispiel werden die numerischen Werte des Attributs *Alter* auf die kategorischen Werte „<=30“, „31…40“ und „>40“ aufgeteilt.

Der Algorithmus geht wie folgt vor:

- Der Baum ist initial ein einzelner, alle Trainingsproben repräsentierender, Knoten (Schritt 1).
- Falls die Trainingsproben alle derselben Klasse angehören wird der Knoten zu einem Blatt, das mit der Klasse beschriftet ist (Schritte 2 und 3).
- Ansonsten benutzt der Algorithmus den so genannten *Informationsgewinn* als Maß für dasjenige Attribut, das die Trainingsproben am besten in Klassen aufteilt. Dieses Attribut wird zum Testattribut des Knotens (Schritte 6 und 7).
- Für jeden bekannten Wert des Testattributs wird ein Ast hinzugefügt und die Trainingsproben werden entsprechend partitioniert (Schritte 8 – 10).
- Der Algorithmus benutzt obige Vorgehensweise rekursiv, um einen Entscheidungsbaum für die Proben jeder Partition zu erzeugen. Dabei braucht ein Attribut das einmal Testattribut eines Knotens war, in den Nachfolgern dieses Knotens nicht mehr als Testattribut berücksichtigt werden (Schritt 13).

- Die rekursive Partitionierung terminiert, sobald eine der folgenden Bedingungen erfüllt ist:
  1. Alle Proben eines Knotens gehören zur selben Klasse (Schritte 2 und 3).
  2. Es sind keine Attribute zum weiteren Partitionieren der Proben mehr übrig (Schritt 5). In diesem Fall wird eine *Mehrheitsentscheidung* durchgeführt (Schritt 6). Dabei wird zunächst der Knoten in ein Blatt verwandelt und dann mit der Klasse gekennzeichnet, die unter den Proben der Partition dieses Knotens am häufigsten vorkommt.
  3. Es gibt keine Proben für den Ast *Testattribut* = a_i (Schritt 11). In diesem Fall wird ein Blatt an diesen Ast angefügt, dessen Klasse durch die am häufigsten vertretene Klasse in der Partition seines Vaters bestimmt ist (Schritt 12).

Algorithmus: *Generiere_Entscheidungsbaum*.
Input: Die Trainingsproben, *Proben*, dargestellt durch kategorische Attribute;
die Menge der Testattributkandidaten, *Attributliste*.
Output: Ein Entscheidungsbaum.

(1) bilde eine Knoten *K*;
(2) If *Proben* gehören alle zur selben Klasse *C* Then
(3) return *K* als Blatt der Klasse *C*;
(4) If *Attributliste* ist leer Then
(5) return *K* als Blatt derjenigen Klasse, die in *Proben* am häufigsten vorkommt;
(6) Wähle *Testattribut* als das Attribut aus *Attributliste* mit dem höchsten *Informationsgewinn*;
(7) Kennzeichne Knoten *K* mit *Testattribut*;
(8) For Each bekannten Wert a_i von *Testattribut*
(9) Füge einen Ast von Knoten *K* mit der Bedingung *Testattribut* = a_i hinzu;
(10) Setze S_i gleich der Menge der Proben aus *Proben* für die *Testattribut* = a_i;
(11) If S_i ist leer Then
(12) Füge ein Blatt gekennzeichnet mit der häufigsten Klasse in *Proben* hinzu;
(13) Else Füge den von
Generiere_Entscheidungsbaum(S_i, *Attributliste – Testattribut*) zurückgegebenen Teilbaum hinzu;

Abbildung 3.2: Der Basisalgorithmus zum Aufbau eines Entscheidungsbaums in Pseudocode.

In Schritt 6 wählt der Algorithmus das Attribut aus *Attributliste* mit dem höchsten Informationsgewinn als Testattribut aus. Dieser *Informationsgewinn* ist ein Auswahlmaß für Attribute, also ein Maß dafür, wie gut der Test auf dieses Attribut die unterschiedlichen Klassen trennt. Das Attribut mit dem höchsten Informationsgewinn minimiert die benötigte Information, um die Proben in den resultierenden Partitionen zu klassifizieren. Ein solcher informationstheoretischer Ansatz minimiert die zu erwartende Anzahl von Tests, um eine unbekannte Probe zu klassifizieren und stellt sicher, dass ein einfacher Baum aufgebaut wird. Im Folgenden wird das mathematische Modell, das dem Maß des Informationsgewinns zugrunde liegt, vorgestellt:

Sei *S* eine Menge von *s* Trainingsproben. Das klassenbestimmende Attribut habe *n* verschiedene Werte, die also *n* verschiedene Klassen K_i ($i = 1,\ldots,n$) definieren. Sei s_i die Anzahl der Proben von *S* aus Klasse K_i. Dann ist die erwartete benötigte Information eine unbekannte Probe zu klassifizieren gegeben durch:

$$I(s_1,...,s_n) = -\sum_{i=1}^{n} p_i \log_2(p_i). \qquad (3.1)$$

Dabei ist p_i die Wahrscheinlichkeit, dass eine beliebige Trainingsprobe zur Klasse K_i gehört. Sie wird abgeschätzt durch s_i/s.

Weiterhin sei *A* ein Attribut mit *v* verschiedenen Werten $\{a_1,\ldots,a_v\}$, die *S* in *v* Teilmengen $\{S_1,\ldots,S_v\}$ aufteilt, wobei S_j alle Trainingsproben mit Wert a_j von *A* enthält. Diese Teilmengen sind genau die Partitionen, die für *A* als Testattribut des Knotens, der *S* enthält, entstehen würden. Sei nun

s_{ij} die Anzahl der Trainingsproben einer Klasse K_i, die auch in S_j enthalten sind. Dann wird die *Entropie von A*, das ist die erwartete benötigte Information für die Partitionierung mit *A* als Testattribut, berechnet durch:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + ... + s_{nj}}{s} I(s_{1j},...,s_{nj}). \quad (3.2)$$

Dabei bedeutet der Bruch in der Summe das Gewicht der Teilmenge S_j, nämlich die Anzahl der Trainingsproben in S_j geteilt durch die gesamte Anzahl aller Trainingsproben. Desto kleiner der Wert für *E(A)* ist, desto größer ist die „Reinheit" der Partitionen.

Nun kann man den *Informationsgewinn*, der sich durch die Verwendung von *A* als Testattribut ergibt, folgendermaßen berechnen:

$$Gewinn(A) = I(s_1,...,s_n) - E(A). \quad (3.3)$$

Gewinn(A) beschreibt also die erwartete Verminderung des Informationsbedarfs, der durch das Wissen um den Wert des Attributs *A* verursacht wird.

Tabelle 3.1: Trainingsproben zur Klassifikation nach *kauft_Computer*

OID	Alter	Einkommen	Student	Kreditwürdigkeit	Klasse: Kauft_Computer
1	<=30	hoch	nein	normal	nein
2	<=30	hoch	nein	ausgezeichnet	nein
3	31…40	hoch	nein	normal	ja
4	>40	mittel	nein	normal	ja
5	>40	niedrig	ja	normal	ja
6	>40	niedrig	ja	ausgezeichnet	nein
7	31…40	niedrig	ja	ausgezeichnet	ja
8	<=30	mittel	nein	normal	nein
9	<=30	niedrig	ja	normal	ja
10	>40	mittel	ja	normal	ja
11	<=30	mittel	ja	ausgezeichnet	ja
12	31…40	mittel	nein	ausgezeichnet	ja
13	31…40	hoch	ja	normal	ja
14	>40	mittel	nein	ausgezeichnet	nein

Beispiel: Tabelle 3.1 enthält eine Menge von Trainingsproben, deren klassenbestimmendes Attribut *kauft_Computer* zwei verschiedene Werte, nämlich „ja" und „nein", annimmt. Es gibt also zwei verschiedene Klassen K_1 (*kauft_Computer* = „ja") und K_2 (*kauft_Computer* = „nein") ($n = 2$). Es gibt 9 Proben der Klasse „ja" und 5 Proben der Klasse „nein". Um den Informationsgewinn für jedes Attribut zu berechnen, muss zunächst der Informationsbedarf für die Klassifikation einer unbekannten Probe mittels Gleichung 3.1 berechnet werden:

$$I(s_1, s_2) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0{,}940.$$

Nun können die den Partitionierungen nach den einzelnen Attributen zugehörigen Entropien mittels Gleichung (3.2) bestimmt werden. Zunächst soll die Entropie für die Partitionierung anhand des Attributs *Alter* bestimmt werden. Dazu muss die Verteilung der Klassenzugehörigkeit der Proben für jeden möglichen Wert von *Alter* bestimmt werden:

Alter = „<=30":

$s_{11} = 2 \qquad s_{21} = 3 \qquad I(s_{11}, s_{21}) = 0{,}971$

Alter = „31…40":

$s_{12} = 4 \qquad s_{22} = 0 \qquad I(s_{12}, s_{22}) = 0$

Alter = „>40“:

$s_{13} = 3 \qquad s_{23} = 2 \qquad I(s_{13}, s_{23}) = 0{,}971$

Dann ergibt sich die erwartete Information, die benötigt wird um eine Probe zu klassifizieren falls nach *Alter* partitioniert wurde, als

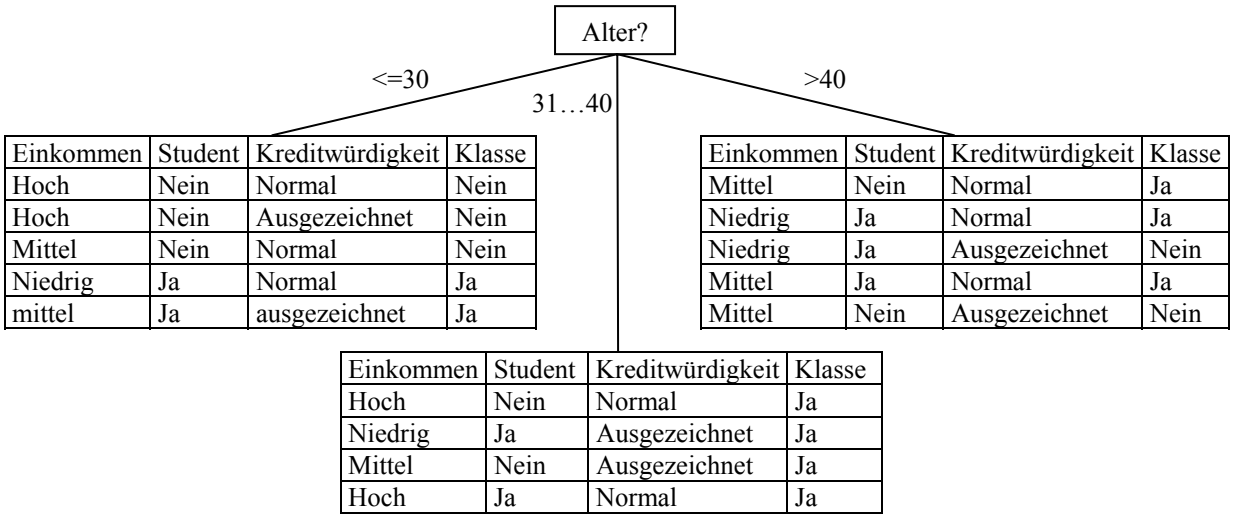
$$E(Alter) = \frac{5}{14} I(s_{11}, s_{21}) + \frac{4}{14} I(s_{12}, s_{22}) + \frac{5}{14} I(s_{13}, s_{23}) = 0{,}694.$$

Also wäre bei einer Partitionierung entsprechend den Werten von *Alter* der Informationsgewinn

$$Gewinn(Alter) = I(s_1, s_2) - E(Alter) = 0{,}246.$$

Analog lassen sich *Gewinn(Einkommen)* = 0,029, *Gewinn(Student)* = 0,151 und *Gewinn(Kreditwürdigkeit)* = 0,048 bestimmen. Damit erhält man für *Alter* den höchsten Informationsgewinn unter allen Attributen und *Alter* wird in Schritt (6) des Basisalgorithmus als *Testattribut* des Wurzelknotens im Entscheidungsbaum ausgewählt.

Nun werden in Schritt (8) und (9) für alle Werte von *Alter*, also „<=30“, „31…40” und „>40”, Äste an den Wurzelknoten angehängt. Diese Äste bekommen dann die entsprechenden Partitionen zugeteilt, wie in Abbildung 3.3 verdeutlicht ist. Es fällt auf, dass alle Proben mit *Alter* = „31…40“ zur selben Klasse, nämlich „ja“ gehören. Beim nächsten rekursiven Aufruf des Algorithmus (Schritt (13)) für diese Partition gehören also alle Proben zur selben Klasse(Schritt (2)), und deswegen wird nun am Ende dieses Astes ein mit „ja“ gekennzeichnetes Blatt angefügt (Schritt (3)). Führt man den Algorithmus fort, so erhält man den Entscheidungsbaum aus Abbildung 3.1.



Einkommen	Student	Kreditwürdigkeit	Klasse
Hoch	Nein	Normal	Nein
Hoch	Nein	Ausgezeichnet	Nein
Mittel	Nein	Normal	Nein
Niedrig	Ja	Normal	Ja
mittel	Ja	ausgezeichnet	Ja

Einkommen	Student	Kreditwürdigkeit	Klasse
Mittel	Nein	Normal	Ja
Niedrig	Ja	Normal	Ja
Niedrig	Ja	Ausgezeichnet	Nein
Mittel	Ja	Normal	Ja
Mittel	Nein	Ausgezeichnet	Nein

Einkommen	Student	Kreditwürdigkeit	Klasse
Hoch	Nein	Normal	Ja
Niedrig	Ja	Ausgezeichnet	Ja
Mittel	Nein	Ausgezeichnet	Ja
Hoch	Ja	Normal	Ja

Abbildung 3.3: Partitionierung nach dem Attribut *Alter*.

3.2 Baumbeschneidung

Beim Aufbau eines Entscheidungsbaums spiegeln viele Äste Anomalien wieder, verursacht durch Ausreißer oder fehlende Attributwerte. Methoden zur Baumbeschneidung (Tree Pruning) beschäftigen sich mit diesem Problem den Baum den Trainingsproben in übertriebener Weise anzupassen. Sie benutzen statistische Maße, um die unzuverlässigsten Äste zu entfernen. Dies führt im Allgemeinen zu einer schnelleren Klassifikation sowie zu einer Verbesserung der Fähigkeit des Baums, unabhängige Testdaten zu klassifizieren. Bei der Funktionsweise der Baumbeschneidung unterscheidet man zwei grundlegende Ansätze, die als Prepruning und Postpruning bekannt sind.

Beim *Prepruning* wird die Erzeugung des Baumes in frühem Stadium angehalten, um voraussichtlich „unreine“ Äste gar nicht erst zu erzeugen und stattdessen direkt ein Blatt in den Baum einzufügen. Dieses Blatt repräsentiert dann die in der zugehörigen Partition am häufigsten vertretene Klasse.

Beim Aufbau des Baums können Maße wie statistische Signifikanz, χ^2, Informationsgewinn und viele andere zum bestimmen der Güte einer Aufteilung herangezogen werden. Falls die Partitionierung der Proben eines Knotens zu einer Aufteilung führt, deren Gütemaß unter einem festgelegten

Schwellwert liegt, so wird eine weitere Partitionierung gestoppt. Die Wahl des richtigen Schwellwertes ist eine schwierige Aufgabe: Ist er zu hoch, so wird der resultierende Baum zu sehr vereinfacht, ist er zu niedrig, so gibt es zu wenig Vereinfachung.

Die zweite Variante der Baumbeschneidung, das *Postpruning*, entfernt Äste von einem voll ausgebildetem Baum. Dabei wird ein Knoten beschnitten, indem seine Äste entfernt werden. Um die zu beschneidenden Knoten auszuwählen, wird für jeden inneren Knoten des Baums eine erwartete Fehlerrate bestimmt, die auftreten würde, falls der Unterbaum des Knotens entfernt würde. Sodann wird die Fehlerrate für den Fall bestimmt, dass der Unterbaum nicht abgeschnitten wird, und zwar aus den Fehlerraten der Söhne, die entsprechend ihrer Verteilung gewichtet sind. Wenn das Beschneiden des Baums zu einer höheren Fehlerrate führt, bleibt der Unterbaum erhalten. Anderenfalls wird der Unterbaum entfernt. Nachdem so sukzessive eine Menge beschnittener Bäume entstanden ist, wird eine unabhängige Testmenge benutzt, um die Genauigkeit jedes Baums zu schätzen. Derjenige Baum mit der größten Genauigkeit wird bevorzugt.

Es ist denkbar Prepruning und Postpruning überlappend in einem kombinierten Ansatz zu nutzen. Postpruning benötigt mehr Rechenaufwand als Prepruning, liefert aber im Allgemeinen einen zuverlässigeren Entscheidungsbaum.

3.3 Skalierbare Algorithmen

Die Leistung der auf dem Basisalgorithmus aufbauenden Algorithmen wie ID3[Qui86], C4.5[Qui93] oder CART[BFOS84] hat sich für relativ kleine Trainingsmengen als sehr gut herausgestellt. Diese Algorithmen sind jedoch nicht skalierbar, d.h. ihre Laufzeit ist nicht proportional zur Größe des Inputs, also der Trainingsmenge. Sobald die Trainingsmenge nicht in den Hauptspeicher passt, müssen diese Algorithmen Trainingsdaten auf den Externspeicher auslagern, was sich erheblich auf die Leistung auswirkt.

Beim Data Mining wird die Entscheidungsbauminduktion typischerweise auf sehr großen Trainingsmengen ausgeführt, wodurch die Notwendigkeit von skalierbaren Algorithmen zur Induktion von Entscheidungsbäumen deutlich wird.

Eine Möglichkeit dieses Problem der Skalierbarkeit zu beheben ist es, die Trainingsmenge in Teilmengen aufzuteilen, und zunächst für jede dieser Teilmengen einen Entscheidungsbaum zu bilden. Der eigentliche Klassifikator wird dann durch die Kombination der einzelnen Entscheidungsbäume gewonnen. Obwohl diese Methode unzweifelhaft die Klassifikation sehr großer Trainingsmengen erlaubt, ist die Genauigkeit eines so gewonnenen Klassifikators nicht so hoch, als wenn er aus der gesamten Trainingsmenge entstanden wäre.

Im Rahmen der Forschung im Bereich des Data Mining sind Algorithmen zur Lösung des Skalierbarkeitsproblems vorgestellt worden. Im Folgenden sollen zwei dieser Algorithmen, nämlich *SLIQ* [AMR96] und *SPRINT* [AMS96], kurz vorgestellt werden.

Unterabschnitt 3.3.1 wird zu diesem Zweck zunächst Gemeinsamkeiten der beiden Algorithmen sowie unterschiedliche Vorgehensweisen in Bezug auf den Basisalgorithmus darstellen. In den darauf folgenden Unterabschnitten 3.3.2 und 3.3.3 werden die Algorithmen SLIQ (3.3.2) und SPRINT (3.3.3) dann im Einzelnen vorgestellt.

3.3.1 Gemeinsamkeiten der Algorithmen SLIQ und SPRINT

Der Basisalgorithmus lässt zur Induktion eines Entscheidungsbaums nur kategorische Attribute zu. Deswegen müssen alle stetigen Attribute kategorisiert werden. SLIQ und SPRINT lassen jedoch auch stetige Attribute ohne vorhergehende Kategorisierung zu. Die Partitionierung anhand eines stetigen Attributs A erfolgt durch das Aufteilen aller Proben der zum aktuellen Knoten gehörenden Partition auf einen *linken* und einen *rechten Sohn*. Dabei bekommt der linke Sohn alle Proben mit $A \leq a$, der rechte Sohn alle Proben mit $A > a$ zugewiesen, wobei a eine im Klassifikationsprozess zu ermittelnde Konstante ist.

Auch kategorische Attribute werden anders partitioniert als beim Basisalgorithmus. Es wird nämlich nicht für alle möglichen Attributwerte ein neuer Sohn erzeugt, sondern wieder nur ein linker und ein

rechter Sohn. Ist A ein kategorisches Attribut, so bekommt der linke Sohn alle Proben mit Attributwerten $A \in S'$, der rechte Sohn alle Proben mit Attributwerten $A \notin S'$ zugewiesen. Dabei ist S' eine nichtleere echte Teilmenge des Wertebereichs von A.

Da die Partitionierung eines Knotens nun immer in zwei Söhnen resultiert und die Proben dieses Knotens somit in zwei Teile „zerschnitten" werden, wird im Folgenden statt von einer Partitionierung von einem *Schnitt* gesprochen.

Der Basisalgorithmus benutzt zur Bestimmung des Testattributs eines Knotens den Informationsgewinn. SLIQ und SPRINT benutzen ein anderes Maß, nämlich den *gini-Index*. Für eine Datenmenge S, die Proben aus n Klassen mit den relativen Häufigkeiten p_j einer Klasse j ($1 \le j \le n$) enthält, ist

$$gini(S) = 1 - \sum p_j \ . \tag{3.4}$$

Für einen Schnitt, der S in zwei Teilmengen S_1 und S_2 mit n_1 bzw. n_2 Proben zerlegt, ist der *Schnittindex* $gini_{split}(S)$ der in S_1 und S_2 aufgeteilten Proben gegeben durch

$$gini_{split}(S) = \frac{n_1}{n} gini(S_1) + \frac{n_2}{n} gini(S_2) \ . \tag{3.5}$$

Um nun den besten Schnitt für einen Knoten herauszufinden, müssen für alle Attribute alle möglichen Schnitte mithilfe des Schnittindexes ausgewertet werden. Derjenige Schnitt mit dem niedrigsten Schnittindex wird ausgewählt.

Bei einem stetigen Attribut A müssen nun also alle Schnitte der Form $A \le a$ mit dem Schnittindex ausgewertet werden. Dazu müssen zunächst alle zum aktuellen Knoten gehörige Proben nach den Werten von A sortiert werden. Als Schnittpunkte können nun jeweils die Mittelwerte m zwischen zwei benachbarten ungleichen Attributwerten verwendet werden, also $A \le m$.

Dieses Sortieren bedeutet einen enorm hohen Aufwand, da es bei jedem Knoten des Baums für alle stetigen Attribute durchgeführt werden muss. Um diesen Aufwand zu vermeiden, benutzen SLIQ und SPRINT die so genannte *Presorting*-Technik. Dazu wird zunächst für jedes Attribut, auch für kategorische, eine *Attributliste* angelegt, und zwar vor Beginn der Induktion des Baums. Diese Attributlisten enthalten bei SLIQ die jeweiligen Attributwerte und eine Proben-Id aller Trainingsproben (RID: Record Identifier) und bei SPRINT zusätzlich den jeweiligen Wert des klassenbestimmenden Attributs.

Durch diese Aufteilung der Attribute auf verschiedene Attributlisten ist es nun möglich das Presorting anzuwenden, welches einfach in dem Sortieren aller Attributlisten stetiger Attribute vor Beginn der Induktion des Baums besteht. Dadurch fällt der Sortieraufwand nur einmal zu Beginn an anstatt bei jedem Knoten wieder, was ohne die Bildung der Attributlisten aufgrund der gegensätzlichen Sortierordnungen der einzelnen Attribute nicht möglich ist.

Bei einem kategorischen Attribut A mit einem Wertebereich *dom*(A) von n verschiedenen Werten kann jede echte nichtleere Teilmenge von *dom*(A) für einen Schnitt verwendet werden. Da die Anzahl aller Teilmengen von *dom*(A) 2^n ist und jeder mögliche Schnitt ausgewertet werden soll, liegt hier ein exponentieller Aufwand vor. Um diesen Aufwand so gering wie möglich zu halten, werden nur dann alle Teilmengen von *dom*(A) ausgewertet, falls n unter einem bestimmten Schwellwert liegt. Ansonsten wird ein schneller Algorithmus aus [NASA92] genutzt, der in [ANR96] kurz skizziert wird. Da die beiden Algorithmen einen besonderen Vorteil aus dem Presorting von stetigen Attributen ziehen, wird im Folgenden nur die Behandlung von stetigen Attributen betrachtet.

Außerdem benutzen die beiden Algorithmen für jeden Knoten des Baums zwei Histogramme *L* und *R*, in denen die relativen Häufigkeiten der verschiedenen Klassen für den linken (L) bzw. den rechten (R) Sohn gespeichert werden. Dadurch kann der jeweilige Schnittindex für jeden Schnitt allein durch die Werte dieser Histogramme berechnet werden.

Tabelle 3.2

RID	Alter	Gehalt	Klasse
1	30	65	G
2	23	15	B
3	40	75	G
4	55	40	B
5	55	100	G
6	45	60	G

Zuletzt ist kurz die von beiden Algorithmen verwendete Baumbeschneidung zu erwähnen. Beide Algorithmen benutzen denselben Postpruning-Algorithmus, der auf dem Prinzip der minimalen Beschreibungslänge (MDL: Minimum Description Length; [Ri89]) beruht. Die Strategie des Algorithmus ist in [AMR96] dargestellt, soll aber hier aus Platzmangel nicht ausgeführt werden. Im Übrigen liegt der Anteil der Baumbeschneidung an der gesamten Zeit, die zur Entscheidungsbauminduktion benötigt wird, nach [AMS96] unter einem Prozent, so dass eine ausschließliche Betrachtung der eigentlichen Induktion des Baums gerechtfertigt ist.

Tabelle 3.2 zeigt Trainingsdaten, die in den Unterabschnitten 3.3.2 und 3.3.3 zur Darstellung verwendet werden.

3.3.2 SLIQ

SLIQ erzeugt aus den Trainingsdaten zunächst Attributlisten für jedes Attribut, die während der Erzeugung dem Presorting entsprechend sortiert werden. Diese Attributlisten werden bei Hauptspeichermangel im Externspeicher gehalten. Zusätzlich zu den Attributlisten wird eine *Klassenliste* erzeugt. In ihr werden für jede Probe der Trainingsdaten die RID, die Klasse und eine Referenz auf denjenigen Knoten im Baum, zu dem die Probe gehört, gespeichert. Diese Klassenliste muss, da der Algorithmus ständig auf sie zugreift, stets im Hauptspeicher gehalten werden.

Abbildung 3.4 zeigt diese Datenstrukturen nach dem Presorting für die Trainingsdaten aus Tabelle 3.2.

Alter	RID
23	2
30	1
40	3
45	6
55	5
55	4

Gehalt	RID
15	2
40	4
60	6
65	1
75	3
100	5

Attributlisten

RID	Klasse	Knoten
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

Klassenliste

Abbildung 3.4

SLIQ baut nun sukzessive die einzelnen Ebenen des Baums auf. Dazu muss für jeden Knoten der aktuellen Ebene zunächst der beste Schnitt gefunden werden. Dazu wird für die aktuelle Ebene jede Attributliste genau einmal durchlaufen. In diesem Durchlauf wird für jeden Eintrag einer Attributliste ein Schnitt bei genau diesem Attributwert ausgewertet. Dafür ist es nötig zu wissen, zu welchem Knoten und zu welcher Klasse dieser Eintrag der Attributliste gehört. Diese Information erhält man aus dem der RID des Eintrags der Attributliste zugehörigen Eintrag in der Klassenliste. Nun können die Histogramme L und R dieses Knotens aktualisiert werden und der Schnittindex kann aus ihren Werten berechnet werden. Für jeden Knoten der aktuellen Ebene wird nun der Schnitt mit dem kleinsten Schnittindex angewendet, es sei denn, der Knoten enthält nur Proben einer Klasse.

Um diesen Schnitt nun wirklich wirksam zu machen, müssen die zugehörigen Knotenreferenzen in der Klassenliste aktualisiert werden. SLIQ erreicht dies, indem er die Attributlisten, deren Attribute tatsächlich für einen Schnitt verwendet wurden, durchläuft und jeweils die Knotenreferenz des korrespondierenden Eintrags der Klassenliste entsprechend dem durchgeführten Schnitt auf den linken bzw. den rechten Sohn ändert.

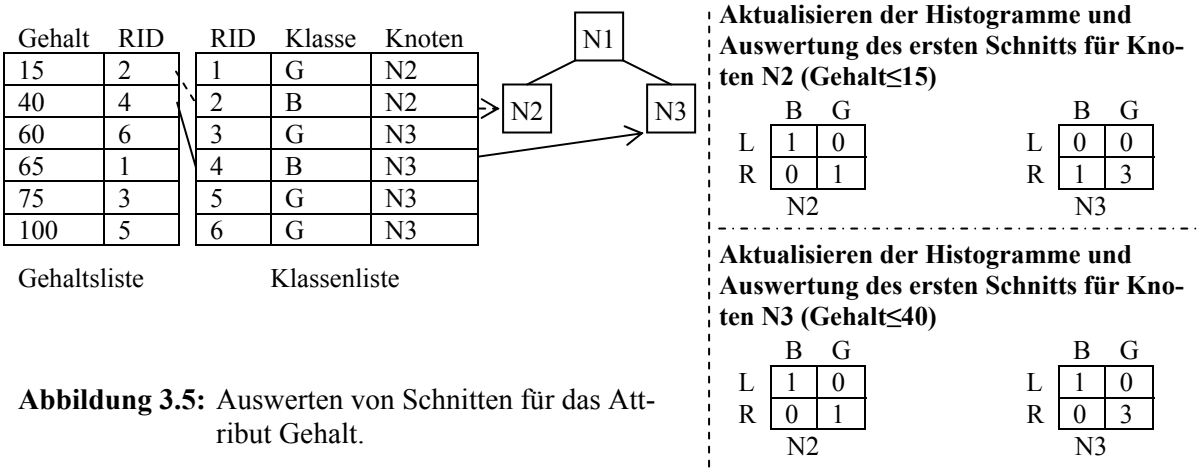


Abbildung 3.5: Auswerten von Schnitten für das Attribut Gehalt.

Abbildung 3.5 stellt das Auswerten der Schnitte für das Attribut *Gehalt* in der zweiten Ebene des Baums dar. Der erste Schnitt in der Wurzel war *Alter* ≤ 35 und wurde vorher schon durchgeführt. Die Histogramme geben die Verteilung der Proben auf die Klassen für jeden Knoten der aktuellen (zwei-

ten) Ebene wieder. Initial gehören alle Proben zum rechten Sohn, was einem Schnitt unter dem Minimum der Attributwerte entspricht (*Gehalt* ≤ 14). Der Wert für L entspricht der relativen Häufigkeit der Klassen von Proben, die das Testprädikat erfüllen, während der Wert für R denjenigen entspricht, die das Testprädikat nicht erfüllen. Der erste Wert in der Gehaltsliste gehört zum Knoten N2. Deshalb wird als erstes ein Schnitt (*Alter* ≤ 15) für den Knoten N2 ausgewertet. Nach diesem Schnitt gehört die zugehörige Probe (*Gehalt* 15, RID 2), die als einzige das Schnittprädikat erfüllt, zum linken Sohn, während der Rest zum rechten Sohn gehört, was sich nach einer Aktualisierung in den Histogrammen L und R widerspiegelt. Als nächstes wird der Schnitt *Gehalt* ≤ 40 für den Knoten N3 ausgewertet, weil der zweite Eintrag der Gehaltsliste zu diesem Knoten gehört. Der aktuelle Eintrag der Attributliste gehört nun zum linken Sohn von N3, der Rest zum rechten Sohn, was sich auch hier nach einer Aktualisierung in den Histogrammen L und R widerspiegelt. Der Rest der Attributwerte wird ebenso durchlaufen. Zusätzlich muss in jedem Schnitt der Schnittindex aus den aktualisierten Histogrammen berechnet werden, um nach Durchlauf aller Attributlisten den besten Schnitt auswählen zu können.

SLIQ ist ein sehr schneller Algorithmus für große Trainingsmengen, dessen Leistung allerdings abnimmt, sobald die Klassenliste nicht mehr in den Hauptspeicher passt. Er ist für kleine Trainingsmengen ebenso gut zu gebrauchen, wie die vorher entwickelten Algorithmen wie z.B. CART oder C4.5), hat jedoch aufgrund seiner erhöhten Skalierbarkeit einen so erheblichen Leistungsvorteil bei großen Trainingsmengen, dass sich ein Vergleich mit den oben erwähnten Algorithmen nicht einmal lohnt.

Die Autoren haben Tests durchgeführt, die zeigen, dass die Laufzeit von SLIQ nahezu linear von der Größe der Trainingsmenge abhängt. Diese Tests wurden auf einer IBM RS/6000 250 Workstation durchgeführt, auf der ein AIX 3.2.5 OS Betriebssystem installiert war. Die Größe des verfügbaren Hauptspeichers betrug 64 MB. Da es keinen Benchmark für Klassifikationsalgorithmen mit großen Datenmengen gibt (für kleine Trainingsmengen wird in der Regel der STATLOG Benchmark [MST94] benutzt, dessen größte Datenmenge jedoch nur 57000 Trainingsproben enthält) haben die Autoren die in [AIS93] vorgestellten synthetischen Datenbanken benutzt, deren Tupel alle neun Attribute haben. Nachdem mit verschieden großen Trainingsmengen Testläufe von SLIQ durchgeführt wurden, wurde ein nahezu linearer Zusammenhang zwischen der Größe der Trainingsmenge und der Laufzeit des Algorithmus festgestellt (bei dieser Systemkonfiguration getestet mit bis zu 10 Millionen Proben).

Tritt allerdings der Fall ein, dass die Klassenliste nicht mehr vollständig in den Hauptspeicher passt, so nimmt, wie schon erwähnt, SLIQ's Leistung ab. Diese Einschränkung der vollständigen Skalierbarkeit behebt der Algorithmus SPRINT, der im nun folgenden Unterabschnitt 3.3.3 beschrieben wird.

3.3.3 SPRINT

SPRINT benutzt keine Klassenliste wie SLIQ, sondern speichert in jeder Attributliste zusätzlich redundant die Klasse der einzelnen Proben. Dies kann geschehen, da die Klasse während der Induktion des Baums nicht geändert wird und somit auch keine Änderungsanomalien zu befürchten sind. Abbildung 3.6 zeigt die Attributlisten von SPRINT für die Trainingsdaten aus Tabelle 3.2 nach dem Presorting.

Alter	Klasse	RID
23	B	2
30	G	1
40	G	3
45	G	6
55	G	5
55	B	4

Attributliste für *Alter*

Gehalt	Klasse	RID
15	B	2
40	B	4
60	G	6
65	G	1
75	G	3
100	G	5

Attributliste für *Gehalt*

Abbildung 3.6: Attributlisten für *Alter* und *Gehalt*

Weiterhin geht durch den Wegfall der Klassenliste auch die Knotenreferenz verloren. Um die Einträge der Attributlisten trotzdem mit dem zugehörigen Knoten zu assoziieren, gehören bei SPRINT zu jedem Knoten Partitionen aller Attributlisten, die den zu diesem Knoten gehörenden Proben entsprechen müssen. Abbildung 3.7 stellt die

Alter	Klasse	RID
23	B	2
30	G	1

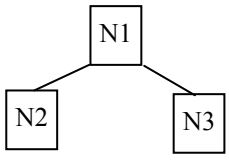


Abbildung 3.7: Partition der Altersliste für N2

Partition der Altersliste für den Knoten N2 dar, falls auf den Knoten N1 der Schnitt Alter ≤ 35 angewendet wird. Die Partitionen werden im Folgenden *Attributlisten eines Knotens* genannt.

Um den besten Schnitt für einen Knoten zu finden, müssen auf allen seine Attributlisten alle möglichen Schnitte ausgewertet werden. Dies erfolgt wie bei SLIQ durch die Nutzung der Histogramme L und R.

Zur Ausführung des Schnittes muss jede Attributliste des Knotens auf seinen linken und rechten Sohn aufgeteilt werden. Bei der Attributliste des Testattributs für den Schnitt ist das einfach: Die Liste wird einmal durchlaufen und jeder Eintrag wird gemäß dem Schnittprädikat in die Attributliste des linken oder des rechten Sohns verschoben.

In den restlichen Attributlisten existiert keine Information über den Attributwert des zum Schnitt benutzten Attributs. Deshalb verläuft die Ausführung der Partitionierung der Attributlisten komplizierter. So wird bei der Partitionierung der Attributliste des Schnittattributs die RID eines jeden Eintrags in eine Hashtabelle eingefügt, die auch die Information enthält, ob die Probe mit dieser RID in den linken oder in den rechten Knoten gehört. Sobald alle RID's in die Hashtabelle eingefügt worden sind, wird beim Durchlauf der restlichen Attributlisten für jeden Eintrag ein Probing mit der jeweiligen RID auf die Hashtabelle durchgeführt, das die Information liefert, zu welchem Sohn der Eintrag gehört. Es erfolgt kein Schnitt, falls alle Proben des Knotens zur selben Klasse gehören.

Auch SPRINT benutzt als Testumgebung eine IBM RS/6000 250 Workstation mit dem Betriebssystem AIX 3.2.5 OS, allerdings nur mit einem verfügbaren Hauptspeicher von 16 MB. Weiterhin wurden ebenfalls die synthetischen Datenbanken aus [AIS93] genutzt.

Abbildung 3.8 zeigt den Zusammenhang zwischen Größe der Trainingsmenge und Ausführungszeit für SLIQ und SPRINT. Es zeigt sich dass SLIQ fast ein lineares Wachstum der Antwortzeit in Abhängigkeit von der Anzahl der Trainingsproben aufweist, allerdings nur bis zu einer Größe von ca. 1,5 Millionen Proben. Dies ist der Punkt, an dem SLIQ's Klassenliste nicht mehr in de Hauptspeicher passt, wodurch ein Seitenflattern entsteht.

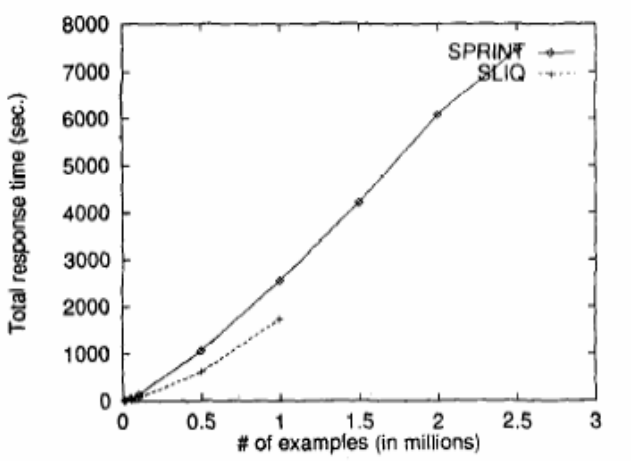


Abbildung 3.8: Zusammenhang zwischen Größe der Trainingsmenge und Ausführungszeit für SLIQ und SPRINT (aus [AMS96])

SPRINT ist für nicht allzugroße Trainingsmengen langsamer als SLIQ. Übersteigt die Größe der Trainingsmenge jedoch einen bestimmten Schwellwert, der von den Autoren auf ca. 1,5 Millionen Proben beziffert wird, so nimmt die Leistung von SLIQ aufgrund des Hauptspeichermangels für die Klassenliste rapide ab. Dahingegen hängt die Leistung von SPRINT immer noch fast linear von der Größe der Trainingsmenge ab, und zwar zumindest bis zu einer Größe von 3 Millionen Proben.

Kapitel 4: Bayes'sche Klassifikation

Die Bayes'sche Klassifikation benutzt statistische Größen, um die Wahrscheinlichkeit, dass eine gegebene Probe zu einer bestimmten vorgegebenen Klasse gehört, vorauszusagen und diese Probe derjenigen Klasse mit der höchsten Wahrscheinlichkeit zuzuweisen. Wie der Name schon sagt, baut die Bayes'sche Klassifikation auf dem Satz von Bayes auf. In Studien wurde ein Bayes'scher Klassifikator entwickelt, bekannt als der naive Bayes'sche Klassifikator, der in der Leistung mit Entscheidungsbäumen und Neuronalen Netzen vergleichbar ist. Bayes'sche Klassifikatoren erreichen bei der Klassifikation außerdem hohe Genauigkeit und Geschwindigkeit insbesondere bei ihrer Anwendung auf große Datenbasen, was sie fürs Data Mining attraktiv macht.

Der *naive Bayes'sche Klassifikator* macht die Annahme, dass der Einfluss eines Attributwertes auf eine gegebene Klasse unabhängig von den Werten anderer Attribute ist. Diese Annahme wird im Allgemeinen *klassenbedingte Unabhängigkeit* genannt. Die klassenbedingte Unabhängigkeit vereinfacht die benötigten Berechnungen und wird in diesem Sinne als naiv bezeichnet.

Die Annahme der klassenbedingten Unabhängigkeit ist im Allgemeinen unrealistisch. So hängt z.B. die Kreditwürdigkeit einer Person in der Regel stark von ihrem Einkommen ab, und somit ist der Einfluss des Attributwertes für die Kreditwürdigkeit auf die Klasse *kauft_Computer=ja* nicht unabhängig von dem Attributwert für das Einkommen. *Bayes'sche Netze* sind graphische Modelle, die die Modellierung solcher Abhängigkeiten zulassen und somit näher an der Realität bleiben.

In Abschnitt 4.1 werden die nötigen stochastischen Vorraussetzungen vermittelt, um dann die Funktionsweisen der naiven Bayes'schen Klassifikation (Abschnitt 4.2) und von Bayes'schen Netzen (Abschnitt 4.3) zu erläutern. Abschnitt 4.4 beschäftigt sich mit dem Trainieren von Bayes'schen Netzen.

4.1 Stochastische Grundkenntnisse

Sei X eine Datenprobe mit unbekannter Klasse und sei H die Annahme, dass X zur Klasse C gehört. Es soll nun die bedingte Wahrscheinlichkeit $P(H|X)$ berechnet werden, um X zu klassifizieren. $P(H|X)$ ist die Wahrscheinlichkeit, dass die Annahme H wahr ist bei gegebener Probe X.

$P(H|X)$ wird die *a-posteriori-Wahrscheinlichkeit von H*, und zwar nachdem X bekannt ist, genannt. Angenommen die Menge der zu klassifizierenden Daten besteht aus Früchten, die durch die Merkmale Farbe und Form beschrieben werden. Weiterhin sei X rot und rund und H sei die Annahme, dass X ein Apfel ist. Dann gibt $P(H|X)$ die Wahrscheinlichkeit wieder, dass X ein Apfel ist, falls X rot und rund ist.

Dagegen ist $P(H)$ die *a-priori-Wahrscheinlichkeit von H*, nämlich die Wahrscheinlichkeit, dass eine beliebige Probe zur Klasse C gehört. In unserem Beispiel wäre das die Wahrscheinlichkeit, dass irgendeine beliebige Frucht ein Apfel ist, unabhängig davon wie diese Frucht aussieht. Die a-posteriori-Wahrscheinlichkeit $P(H|X)$ benutzt also mehr Information, ähnlich einem gewissen Hintergrundwissen, als die a-priori-Wahrscheinlichkeit $P(H)$, und zwar die Information über die Eigenschaften von X.

Analog ist $P(X|H)$ die *a-posteriori-Wahrscheinlichkeit von X* gegeben H. Das ist im Beispiel die Wahrscheinlichkeit, dass X rot und rund ist, und zwar wenn schon bekannt ist, dass X ein Apfel ist. $P(X)$ dagegen ist die *a-priori-Wahrscheinlichkeit von X*, also die Wahrscheinlichkeit, dass eine beliebige Probe die durch X gegebenen Eigenschaften hat. Im Beispiel ist das die Wahrscheinlichkeit, dass eine beliebige Probe aus der Menge der Früchte rot und rund ist.

Die Wahrscheinlichkeiten $P(X)$, $P(H)$ und $P(X|H)$ können unter zu Hilfenahme der Trainingsmenge abgeschätzt werden. Zur Klassifikation wird jedoch die a-posteriori-Wahrscheinlichkeit $P(H|X)$ von H gegeben X benötigt. Hier kommt nun der Satz von Bayes zum Tragen, der es ermöglicht, $P(H|X)$ aus $P(X)$, $P(H)$ und $P(X|H)$ zu berechnen. Er kann durch folgende Formel ausgedrückt werden:

$$P(H \mid X) = \frac{P(X \mid H)P(H)}{P(X)}. \qquad (4.1)$$

Im folgenden Abschnitt 4.2 wird dargestellt, wie der Satz von Bayes zur naiven Bayes'schen Klassifikation genutzt wird.

4.2 Naive Bayes'sche Klassifikation

Dieser Abschnitt wird die Funktionsweise des naiven Bayes'schen Klassifikators erläutern. Schritthaltend mit der Erläuterung soll die Klassifikation mit dem naiven Bayes'schen Klassifikator anhand eines Beispiels dargestellt werden. Die Trainingsproben für das Beispiel sind dieselben, die zur Erläuterung der Entscheidungsbauminduktion genutzt wurden, und sind in Tabelle 3.1 zu finden. Die naive Bayes'sche Klassifikation verfährt wie folgt:

1. Jede Datenprobe ist dargestellt durch einen *n-dimensionalen Eigenschaftsvektor* $X = (x_1,\ldots,x_n)$, der die n Messwerte für die Probe enthält, die den Attributen $A_1,\ldots,A_n$ entsprechen.

Beispiel: Die Datenproben sind durch die Attribute A_1 = *Alter*, A_2 = *Einkommen*, A_3 = *Student* und A_4 = *Kreditwürdigkeit* beschrieben. Die unbekannte zu klassifizierende Datenprobe ist durch

X = („<=30“, „mittel“, „ja“, „normal“)

gegeben.

2. Es gebe m Klassen $C_1,\ldots,C_m$. Der naive Bayes'sche Klassifikator wird für jede unbekannte Probe vorhersagen, dass sie zu der Klasse mit der *höchsten a-posteriori- Wahrscheinlichkeit* gegeben X gehört. Er ordnet eine unbekannte Datenprobe X also der Klasse C_i genau dann zu, wenn

$$P(C_i \mid X) > P(C_j \mid X) \text{ für alle } 1 \le j \le m, j \ne i.$$

$P(C_i|X)$ muss also maximiert werden. Nach dem Satz von Bayes ist

$$P(C_i \mid X) = \frac{P(X \mid C_i)P(C_i)}{P(X)}. \qquad (4.2)$$

Beispiel: Es gibt die beiden Klassen C_1 (*Kauft_Computer* = „ja“) und C_2 (*kauft_Computer* = „nein“).

3. $P(X)$ ist für die verschiedenen Klassen konstant, es muss also nur noch $P(X|C_i)P(C_i)$ *maximiert* werden. Falls die a-priori-Wahrscheinlichkeiten für die C_i's nicht bekannt sind, wird für gewöhnlich angenommen, dass die einzelnen Klassen die gleichen Wahrscheinlichkeiten haben, also $P(C_1) = \ldots = P(C_m) = 1/m$. In diesem Fall reicht es aus, $P(X|C_i)$ zu maximieren. Die einzelnen a-priori-Wahrscheinlichkeiten können jedoch, anstatt als gleich angenommen zu werden, auch durch ihre relativen Häufigkeiten in der Trainingsmenge abgeschätzt werden, also $P(C_i) = s_i/s$. Dabei ist s_i die Anzahl der Trainingsproben aus Klasse C_i, während s die gesamte Anzahl der Trainingsproben ist.

Beispiel: Die a-priori-Wahrscheinlichkeiten der einzelnen Klassen sind $P(C_1)$ = $P(kauft\_Computer = „ja“) = s_1/s = 9/14 = 0{,}643$ und $P(C_2) = P(kauft\_Computer = „nein“) = s_2/s = 5/14 = 0{,}357$.

4. Bei Datensätzen mit vielen Attributen ist es höchst aufwendig $P(X|C_i)$ zu berechnen. Um die Berechnung zu vereinfachen, wird bei der naiven Bayes'schen Klassifikation wie bereits erwähnt die Annahme der *klassenbedingten Unabhängigkeit* gemacht. Das bedeutet, dass die Werte der einzelnen Attribute bei gegebener Klasse unabhängig voneinander sind und somit keine Abhängigkeitsbeziehungen unter den Attributen existieren. Dann kann die a-posteriori-Wahrscheinlichkeit von X gegeben C_i als

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) \qquad (4.3)$$

berechnet werden.

5. Die bedingten Wahrscheinlichkeiten $P(x_k|C_i)$ entsprechen der Wahrscheinlichkeit, dass Attribut A_k den Wert x_k annimmt, falls X zur Klasse C_i gehört. Ihre Abschätzung verläuft für kategorische und stetige Attribute verschieden:

a) Falls A_k ein *kategorisches Attribut* ist, wird $P(x_k|C_i)$ abgeschätzt durch s_{ik}/s_i. s_{ik} steht für die Anzahl derjenigen Proben aus Klasse C_i, für die das Attribut A_k den Wert x_k annimmt, und s_i ist die Anzahl der Proben aus Klasse C_i.

Beispiel: Die einzelnen a-posteriori-Wahrscheinlichkeiten der x_k werden abgeschätzt durch:

$P(x_1|C_1) = P(„<=30“, kauft\_Computer=„ja“) = s_{11}/s_1 = 2/9 = 0{,}222$,
$P(x_2|C_1) = P(„mittel“, kauft\_Computer=„ja“) = s_{12}/s_1 = 4/9 = 0{,}444$,
$P(x_3|C_1) = P(„ja“, kauft\_Computer=„ja“) = s_{13}/s_1 = 6/9 = 0{,}667$,
$P(x_4|C_1) = P(„normal“, kauft\_Computer=„ja“) = s_{14}/s_1 = 6/9 = 0{,}667$,
$P(x_1|C_2) = P(„<=30“, kauft\_Computer=„nein“) = s_{21}/s_2 = 3/5 = 0{,}600$,
$P(x_2|C_2) = P(„mittel“, kauft\_Computer=„nein“) = s_{22}/s_2 = 2/5 = 0{,}400$,
$P(x_3|C_2) = P(„ja“, kauft\_Computer=„nein“) = s_{23}/s_2 = 1/5 = 0{,}200$,
$P(x_4|C_2) = P(„normal“, kauft\_Computer=„nein“) = s_{24}/s_2 = 2/5 = 0{,}400$.

Daraus ergeben sich die a-posteriori-Wahrscheinlichkeiten für X als

$P(X|C_1) = P(x_1|C_1)*P(x_2|C_1)*P(x_3|C_1)*P(x_4|C_1)= 0{,}222*0{,}444*0{,}667*0{,}667 = 0{,}044$

und

$P(X|C_2) = P(x_1|C_2)*P(x_2|C_2)*P(x_3|C_2)*P(x_4|C_2) = 0{,}600*0{,}400*0{,}200*0{,}400 = 0{,}019$.
Falls nun die a-priori-Wahrscheinlichkeiten $P(C_1)$ und $P(C_2)$ gleich wären, also $P(C_1) = P(C_2)$, so wäre die Klassifikation hier vollständig, da die einzelnen $P(C_i)$ bei Gleichheit keinen Einfluss auf die Maximierung von $P(X|C_i)P(C_i)$ haben. X würde also der Klasse C_1 zugeordnet werden. Bei ungleichen $P(C_i)$, wie es hier der Fall ist, fällt die endgültige Klassifikationsentscheidung jedoch erst im sechsten Schritt.

b) Falls A_k jedoch ein *stetiges Attribut* ist, wird in der Regel eine Gauß-Verteilung zugrunde gelegt. Dann ist

$$P(x_k \mid C_i) = \frac{1}{\sqrt{2\pi}\sigma_{C_i}} e^{-\frac{(x_k - \mu_{C_i})^2}{2\sigma_{C_i}^2}}, \qquad (4.4)$$

wobei μ_{Ci} und σ_{Ci} die aus den Trainingsproben der Klasse C_i ermittelten Werte für Mittelwert und Standardabweichung bezeichnen. Die Gauß-Verteilung hat sich in der Statistik als gute Approximation für jegliche Wahrscheinlichkeitsverteilungen erwiesen.

6. Die Klassifikation einer unbekannten Probe X involviert dann die Berechnung von $P(X|C_i)P(C_i)$ für jede Klasse C_i, worauf X der Klasse C_i zugeordnet wird genau dann wenn

$$P(X \mid C_i)P(C_i) > P(X \mid C_j)P(C_j) \text{ für alle } 1 \le j \le m, j \ne i.$$

X wird also derjenigen Klasse C_i zugeteilt, für die $P(X|C_i)P(C_i)$ maximal ist.
Beispiel: Es ergeben sich folgende $P(X|C_i)P(C_i)$:
$P(X|C_1)P(C_1) = 0{,}044*0{,}643 = 0{,}028$ und
$P(X|C_2)P(C_2) = 0{,}019*0{,}357 = 0{,}007$.
Da $P(X|C_i)P(C_i)$ für C_1 maximal ist, ordnet der naive Bayes'sche Klassifikator die Probe X der Klasse C_1, also *kauft_Computer* = „ja", zu.

In der Theorie haben Bayes'sche Klassifikatoren die kleinste Fehlerrate im Vergleich mit allen anderen Klassifikatoren, da sie sich durch die Nutzung des statistischen Modells am genauesten auf die Trainingsmenge einstellen. Allerdings ist dies in der Praxis nicht immer der Fall. Dies ergibt sich einerseits aus der unrealistischen Annahme der klassenbedingten Unabhängigkeit sowie andererseits aus dem Mangel an verfügbaren Daten über die verwendeten Wahrscheinlichkeiten. So kann bei stetigen Attributen eine Gauß-Verteilung, oder auch jede andere Verteilung, nie eine genaue Aussage über die Wahrscheinlichkeit eines gegebenen Attributwertes machen. Nichtsdestotrotz kann der *naive Bayes'sche Klassifikator* in vielen Beziehungen im Vergleich mit Entscheidungsbäumen und Neuronalen Netzen mithalten.

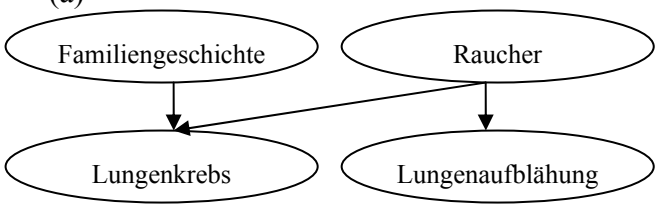
4.3 Bayes'sche Netze

Die naive Bayes'sche Klassifikation macht die Annahme der klassenbedingten Unabhängigkeit, um den Berechnungsaufwand einzuschränken. Wenn diese Annahme sich als richtig erweist, gewinnt der naive Bayes'sche Klassifikator ohne Zweifel jeden Vergleich in Bezug auf Genauigkeit und Geschwindigkeit mit anderen Klassifikatoren. Allerdings existieren in der Realität zumeist Abhängigkeitsbeziehungen zwischen einzelnen Attributen.

Bayes'sche Netze erlauben es diese Abhängigkeiten graphisch zu modellieren. Sie spezifizieren bedingte Verbundwahrscheinlichkeitsverteilungen. Sie erlauben die Definition von klassenbedingten Unabhängigkeit zwischen Teilmengen von Attributen. Außerdem bieten sie ein graphisches Modell kausaler Beziehungen, auf dessen Basis Lernalgorithmen ausgeführt werden können. Diese Netze sind auch als *Belief Netze* oder *Probabilistische Netze* bekannt. Hier wird allerdings die Bezeichnung Bayes'sche Netze beibehalten.

Ein Bayes'sches Netz besteht aus zwei Teilen. Der erste Teil ist ein *gerichteter azyklischer Graph*, dessen Knoten Zufallsvariablen (hier Attribute von Datenmengen) und dessen Kanten probabilistische Abhängigkeiten darstellen. Für eine Kante von einem Knoten A zu einem Knoten B heißt A der *Vater* oder der *direkte Vorgänger* von B, und B ist der *Nachfolger* von A. Jedes Attribut ist bei gegebenen direkten Vorgängern unabhängig von seinen Nicht-Nachfolgern in dem Graph.

(a)



(b)

	FG,R	FG,~R	~FG,R	~FG,~R
LK	0,8	0,5	0,7	0,1
~LK	0,2	0,5	0,3	0,9

Abbildung 4.1: (a) Ein einfaches Bayes'sches Netz. (b) CPT für die Werte des Attributes *Lungenkrebs* (LK) mit jeder möglichen Kombination der Vorgängerknoten *Familiengeschichte* (FG) und *Raucher* (R). Abbildung modifiziert aus [RN95].

Abbildung 4.1(a) zeigt ein einfaches Bayes'sches Netz für vier Boole'sche Attribute, das aus [RN95] modifiziert übernommen wurde. Die Kanten repräsentieren kausales Wissen. So wird zum Beispiel die Wahrscheinlichkeit, dass eine Person *Lungenkrebs* hat, von der *Familiengeschichte* der Lungenkrebsfälle sowie von der Tatsache, ob die Person *Raucher* ist oder nicht, beeinflusst. Außerdem zeigt der Graph, dass die Wahrscheinlichkeit für einen Lungenkrebsfall unabhängig von einer eventuellen *Lungenaufblähung* ist, falls die direkten Vorgänger von *Lungenkrebs*, also *Familiengeschichte* und *Raucher* gegeben sind. Das bedeutet, sobald die Attribute *Familiengeschichte* und *Raucher* bekannt sind, bietet das Wissen um den Wert des Attributes *Lungenaufblähung* keine zusätzliche Information hinsichtlich des Wertes für *Lungenkrebs*.

Der zweite Teil eines Bayes'schen Netzes besteht aus je einer Tabelle mit bedingten Wahrscheinlichkeiten (CPT: Conditional Probability Table) für jedes Attribut. Die CPT für ein Attribut A hält die bedingten Wahrscheinlichkeiten *P(A|Vorgänger(A))* fest, wobei *Vorgänger*(A) für alle direkten Vorgänger von A im Graph steht. Abbildung 4.1(b) zeigt eine CPT für das Attribut *Lungenkrebs*. Dabei ist die bedingte Wahrscheinlichkeit für jeden Wert von *Lungenkrebs* für alle Kombinationen von möglichen Werten der Vorgänger angegeben. Für die Einträge links oben und rechts unten ergeben sich dann die bedingten Wahrscheinlichkeiten

P(*Lungenkrebs* = „Ja" | *Familiengeschichte* = „Ja", *Raucher* = „Ja") = 0,8 und
P(*Lungenkrebs* = „Nein" | *Familiengeschichte* = „Nein", *Raucher* = „Nein") = 0,9.

Diese Wahrscheinlichkeiten P*(A|Vorgänger(A))* muss dass Netz erlernen (siehe Abschnitt 4.4).

Die Verbundwahrscheinlichkeit einer Probe $(x_1,\ldots,x_n)$, deren Werte den Attributen $A_1,\ldots,A_n$ entsprechen, wird durch

$$P(x_1,...,x_n) = \prod_{i=1}^{n} P(x_i \mid Vorgänger(A_i)) \qquad (4.5)$$

berechnet, wobei die Werte für $P(x_i|Vorgänger(A_i))$ den Einträgen der CPT für A_i entsprechen.

Ein Knoten des Netzes kann als „Ausgabeknoten" ausgewählt werden und so ein klassenbestimmende Attribut repräsentieren. Um nun eine Probe $X = (x_1,\ldots,x_n)$ zu klassifizieren, genügt es die Spalte der CPT des Ausgabeknotens mit den zur Probe passenden Werten der Vorgänger des Ausgabeknotens zu betrachten. Der Klassifikationsprozess kann diese Spalte als Wahrscheinlichkeitsverteilung für das klassenbestimmende Attribut ausgeben, also die a-posteriori-Wahrscheinlichkeiten der einzelnen Klassen gegeben einer Probe X. Es ist auch möglich, dass es mehr als einen Ausgabeknoten gibt.

Auf ein so beschriebenes Bayes'sches Netz können nun Algorithmen zum Lernen angewendet werden. Der folgende Abschnitt stellt einen solchen Algorithmus zum Trainieren von Bayes'schen Netzen vor.

4.4 Trainieren von Bayes'schen Netzen

Beim Anlernen oder Trainieren von Bayes'schen Netzen sind verschiedene Szenarien möglich, die Einfluss auf die Methodik des Lernens haben. Zum einen spielt es eine Rolle, ob die Struktur des Netzwerkes vorgegeben ist, oder ob diese erst aus den Trainingsproben ermittelt werden muss. Außerdem ist wichtig, ob die Attribute des Netzes sichtbar oder versteckt in allen oder einigen Trainingsproben sind.

Falls die Netzwerkstruktur vorgegeben ist und alle Attribute für alle Proben sichtbar sind, ist das Anlernen eines Bayes'schen Netzes direkt erreichbar, indem die CPT-Einträge ähnlich wie die a-posteriori-Wahrscheinlichkeiten berechnet werden.

Bei bekannter Netzwerkstruktur in Verbindung mit einigen versteckten Attributen für mehrere Trainingsproben kann eine Methode des Gradientenabstiegs genutzt werden, um das Bayes'sche Netz zu trainieren. Das Ziel ist dabei, die Werte für die einzelnen CPT-Einträge zu erlernen.

Sei nun S die Menge der s Trainingsproben $X_{1,\ldots,}X_s$ und sei w_{ijk} der CPT-Eintrag für das Attribut $A_i = a_{ij}$ mit den direkten Vorgängern $U_i = u_{ik}$. Für den Eintrag w_{ijk} links oben in Abbildung 4.1(b) ist A_i das Attribut *Lungenkrebs* mit dem Wert a_i = „ja". U_i listet die direkten Vorgänger von A_i, nämlich {*Familiengeschichte*, *Raucher*}, während u_i die zu U_i gehörigen Werte {„ja", „ja"} enthält.

Die w_{ijk} werden als Gewichte dargestellt. Die Menge aller Gewichte wird mit w bezeichnet. Den einzelnen Gewichten sind anfangs zufällige Wahrscheinlichkeitswerte zugeordnet. Die Strategie des Gradientenabstiegs führt einen mit dem Bergsteigen vergleichbaren Greedy-Algorithmus durch, indem in jeder Iteration die Gewichte aktualisiert werden, um diese zu einem lokalen Optimum zu führen.

Die Methode sucht nach denjenigen Werten w_{ijk}, die die Trainingsdaten am besten modellieren, basierend auf der Annahme, dass zunächst alle möglichen Zusammensetzungen von w dieselben Wahrscheinlichkeiten haben. Das Ziel ist somit die Maximierung von

$$P_w(S) = \prod_{d=1}^{s} P_w(X_d).$$

Um dieses Ziel zu erreichen ist es sinnvoll, dem Gradienten von $ln\ P_w(S)$ zu folgen, um das Problem zu vereinfachen. Bei gegebener Netzwerkstruktur und initialen w_{ijk} verfährt der Algorithmus in jeder Iteration wie folgt:

1. Zunächst müssen für alle i, j, k die Gradienten berechnet werden:

$$\frac{\delta \ln P_w(S)}{\delta w_{ijk}} = \sum_{d=1}^{s} \frac{P(Y_i = y_{ij}, U_i = u_{ik} \mid X_d)}{w_{ijk}} \qquad (4.6)$$

   Die Wahrscheinlichkeit auf der rechten Seite von Gleichung (4.6) muss für jede Trainingsprobe X_d in S berechnet werden. Sie sei der Einfachheit halber mit p gekennzeichnet. Falls die von Y_i und U_i repräsentierten Attribute für einige X_d versteckt sind, d.h. falls die entsprechende Attributwerte für X_d nicht existieren, kann die entsprechende Wahrscheinlichkeit p aus den sichtbaren Attributen der Probe mithilfe von Standardalgorithmen für die Inferenz von Bayes'schen Netzen ermittelt werden (siehe z.B. *http://www.hugin.dk*).

2. Es werden neue Gewichte w_{ijk} berechnet, also für alle i, j, k:

$$w_{ijk} = w_{ijk} + l \frac{\delta \ln P_w(S)}{\delta w_{ijk}}. \qquad (4.7)$$

   Dabei ist l die Lernrate, die die Feinheit der Lernschritte festlegt. Sie wird auf eine kleine Konstante gesetzt.

3. Da die w_{ijk}'s Wahrscheinlichkeitswerte sind, müssen ihre Werte zwischen *0,0* und *1,0* liegen, und die Summe $\sum_j w_{ijk}$ muss gleich eins sein für alle i, k. Um dies zu erreichen, müssen die Gewichte nach ihrer Aktualisierung entsprechend Gleichung (4.7) renormalisiert werden.

Es gibt auch mehrere Algorithmen, die die Netzwerkstruktur eines Bayes'schen Netzes aus den Trainingsproben erlernen, falls die Attribute sichtbar sind. Dieses Problem gehört der diskreten Optimierung an und kann hier nicht näher behandelt werden, da es den ohnehin knappen Rahmen dieser Ausarbeitung vollends sprengen würde.

Kapitel 5: Prädiktion mittels Regression

Die beiden vorangegangenen Kapitel haben das Problem der Klassifikation behandelt, also das Vorraussagen des Wertes eines Attributs mit kategorischem Wertebereich. In diesem Kapitel sollen nun einige Lösungsansätze der *Regression* zur Vorhersage des Wertes eines stetigen Attributs, also des *Prädiktionsproblems*, vorgestellt werden. Regression beschäftigt sich mit der Zurückführung von neuen Daten auf die Verteilung einer analysierten Trainingsmenge.

Erstaunlich viele Prädiktionsprobleme können durch die *lineare Regression* gelöst werden, die in Abschnitt 5.1 inklusive der *multiplen Regression* diskutiert wird. Abschnitt 5.2 wird sich mit der Erweiterung des linearen Regressionsmodells zur *nichtlinearen Regression* beschäftigen. Da dieser Ausarbeitung der Platz für eine detaillierte Darstellung fehlt, soll in den folgenden Abschnitten eine eher intuitive Darstellung der oben genannten Konzepte erfolgen.

5.1 Lineare und Multiple Regression

Die *lineare Regression* modelliert Daten durch eine lineare Funktion. Sie ist die einfachste Form der Regression und wird gerade um dieser Einfachheit willen häufig verwendet. Die einfache lineare Regression stellt eine Zufallsvariable Y, die zu ermittelnde *Antwortvariable*, mithilfe einer auf eine andere Zufallsvariable X, die *Schätzervariable*, angewendeten linearen Funktion dar. Die Berechnung von Y findet also mithilfe folgender *Regressionsgleichung* statt:

$$Y = \alpha + \beta X. \tag{5.1}$$

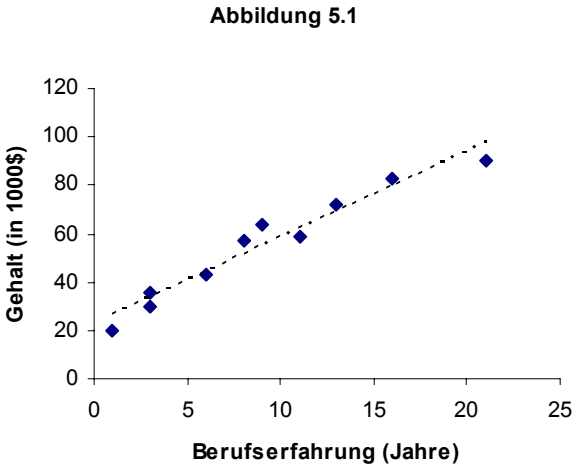
Dabei wird die Varianz von Y als konstant angenommen. α und β werden als Regressionskoeffizienten bezeichnet und geben den Y-Achsenabschnitt und die Steigung der Funktion an. Die Schwierigkeit ist nun die Bestimmung dieser Regressionskoeffizienten.

Die populärste Methode, dies zu erreichen, ist die *Methode der kleinsten Quadrate*. Diese minimiert den Fehler, der zwischen den existierenden Datenproben und der zu bestimmenden Funktion besteht. Sind s Proben der Form (x_1, y_1), …, (x_s, y_s) gegeben, dann werden die Regressionskoeffizienten bei der Methode der kleinsten Quadrate unter Benutzung folgender Gleichungen bestimmt:

$$\beta = \frac{\sum_{i=1}^{s}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s}(x_i - \bar{x})^2} \tag{5.2}$$

$$\alpha = \bar{y} - \beta\bar{x} \tag{5.3}$$

Hierbei stellt $\bar{x}$ den Mittelwert von $x_1,\ldots,x_s$ dar , ebenso wie $\bar{y}$ den Mittelwert von $y_1,\ldots,y_s$ bezeichnet. Häufig gewinnt man durch die Regressionskoeffizienten α und β gute Approximationen von ansonsten komplizierten Regressionsgleichungen.



X – Berufserfahrung (Jahre)	Y - Gehalt (in 1000$)
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

Tabelle 5.1: Gehaltsdaten

Beispiel 5.1: Tabelle 5.1 zeigt eine Menge von Trainingsproben, bei denen X die in Jahren angegebene Berufserfahrung eines Studienabsolventen und Y das Jahresgehalt eines solchen darstellen. Abbildung 5.1 zeigt eine graphische Darstellung dieser Proben, die einen linearen Zusammenhang zwischen X und Y suggeriert. Nun kann der Zusammenhang zwischen dem Gehalt einer Person und ihrer Berufserfahrung durch Gleichung (5.1) modelliert werden.

Mit den gegebenen Proben können die Mittelwerte $\bar{x} = 9{,}1$ und $\bar{y} = 55{,}4$ ermittelt werden. Setzt man diese Werte in die Gleichungen (5.2) und (5.3) ein, so erhält man die Regressionskoeffizienten

$$\beta = \frac{(3-9{,}1)(30-55{,}4)+(8-9{,}1)(57-55{,}4)+...+(16-9{,}1)(83-55{,}4)}{(3-9{,}1)^2+(8-9{,}1)^2+...+(16-9{,}1)^2} = 3{,}5 \text{ und}$$

$$\alpha = 55{,}4-(3{,}5)(9{,}1) = 23{,}6.$$

So wird das Gehalt eines Studienabsolventen in Abhängigkeit seiner Berufserfahrung basierend auf den Datenproben aus Tabelle 5.1 nach der Methode der kleinsten Quadrate also durch $Y = 23{,}6 + 3{,}5X$ abgeschätzt. Somit würde das Gehalt eines Studienabsolventen mit 10 Jahren Berufserfahrung also auf 58600 $ geschätzt werden.

Die multiple Regression ist eine Erweiterung der einfachen linearen Regression. Sie lässt mehr als eine Schätzervariable zu und erlaubt somit die Modellierung einer Antwortvariablen Y durch die lineare Funktion eines mehrdimensionalen Attributvektors. Gleichung (5.4) zeigt ein multiples Regressionsmodell mit zwei Schätzervariablen X_1 und X_2:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2. \qquad (5.4)$$

Die Methode der kleinsten Quadrate lässt sich auch auf multiple Regressionsmodelle anwenden, die Berechnung der Regressionskoeffizienten ist jedoch erheblich aufwendiger als bei der einfachen linearen Regression, so dass hier auf eine Darstellung verzichtet werden muss.

5.2 Nichtlineare Regression

Die lineare Regression eröffnet die Möglichkeit, lineare Zusammenhänge zwischen Antwortvariable und Schätzervariablen durch Regressionskoeffizienten auszudrücken. Es sind aber auch andere, z.B. polynomiale Zusammenhänge denkbar. *Polynomiale Regression* wird durch das Hinzufügen polynomialer Terme zu dem linearen Modell modelliert. Sodann können die nichtlinearen Variablen durch lineare ersetzt werden und das nichtlineare Modell kann in ein lineares Modell, das mit der Methode der kleinsten Quadrate gelöst werden kann, überführt werden.

Beispiel 5.2: Es sei folgende polynomiale Beziehung zwischen der Antwortvariablen Y und der Schätzervariablen X gegeben:

$$Y = \alpha + \beta_1 X + \beta_2 X^2 + \beta_3 X^3. \qquad (5.5)$$

Um diese Gleichung in eine lineare zu überführen, müssen neue Variablen definiert werden:

$$X_1 = X, \qquad X_2 = X^2, \qquad X_3 = X^3.$$

Die so erhaltenen Variablen können nun in das lineare Modell aus Gleichung (5.5) eingesetzt werden und man erhält folgendes, durch die Methode der kleinsten Quadrate lösbares, lineare Modell:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3. \qquad (5.6)$$

Natürlich gibt es auch nichtlineare Modelle, die sich nicht in lineare Modelle überführen lassen, wie z.B. die Summe von exponentiellen Termen. In diesen Fällen müssen wesentlich aufwendigere Berechnungen basierend auf höchst komplexen Formeln durchgeführt werden, um zu einem Schätzwert nach der Methode der kleinsten Quadrate zu gelangen.

Kapitel 6: Zusammenfassung

Der Zweck dieser Ausarbeitung ist es, dem Leser ein tieferes Verständnis der Konzepte der Klassifikation und Prädiktion im Rahmen des Data Mining zu vermitteln. In dieser Zusammenfassung soll der Inhalt der Ausarbeitung noch einmal kurz aufgearbeitet werden und dem Leser eine Vorstellung gegeben werden, was er beim Lesen gelernt hat:

Zunächst wurden in Kapitel 2 das grundlegende Konzept des *Klassifikationsprozesses* vorgestellt. Dieser besteht aus zwei Schritten: Im ersten Schritt wird ein *Klassifikator* in Form von *Klassifikationsregeln* durch die Analyse der Menge der Trainingsproben mithilfe von *überwachtem Lernen* erzeugt, während der Klassifikator im zweiten Schritt bei genügender Genauigkeit auf *unbekannte Proben* angewendet werden kann, um diese zu klassifizieren.

In Kapitel 3 wurde die Nutzung von Entscheidungsbäumen als Klassifikator erläutert. Der Vorteil von Entscheidungsbäumen ist, dass sie für den Menschen intuitiv verständlich sind und dass sie leicht in Klassifikationsregeln umgesetzt werden können. Es wurde ein *Basisalgorithmus* vorgestellt, der in der Lage ist einen als Klassifikator nutzbaren Entscheidungsbaum zu erzeugen. Weiterhin wurde die Möglichkeit der *Baumbeschneidung* angesprochen, um die Größe des resultierenden Entscheidungsbaums und somit auch die Klassifikationszeit zu verringern. Sodann wurde auf die beschränkte Nutzbarkeit des Basisalgorithmus und ähnlicher Algorithmen hingewiesen. Diese beschränkte Nutzbarkeit ergibt sich durch die fehlende Skalierbarkeit dieser Algorithmen. Deshalb wurden als nächstes die Algorithmen SLIQ und SPRINT vorgestellt, die das Problem der Skalierbarkeit beheben und speziell für das Data Mining entwickelt wurden. Dabei ist SLIQ ein sehr schneller Algorithmus mit eingeschränkter Skalierbarkeit, während SPRINT nicht ganz so schnell, dafür aber vollständig skalierbar ist.

Kapitel 4 beschäftigt sich mit der Nutzung des *Satzes von Bayes* zur Klassifikation. Es wurde zunächst der *naive Bayes'sche Klassifikator* vorgestellt. Dieser macht zwar aufgrund einer Vereinfachung der Berechnung die im Allgemeinen unrealistische Annahme der *klassenbedingten Unabhängigkeit*, kann sich jedoch durchaus mit den Entscheidungsbäumen vergleichen. *Bayes'sche Netze* vermeiden die Annahme der klassenbedingten Unabhängigkeit, indem sie die explizite Modellierung von Abhängigkeiten zwischen einzelnen Attributen ermöglichen. Bayes'sche Netze bestehen aus zwei Teilen: einem azyklischen gerichteten Graph, der Abhängigkeiten zwischen Attributen modelliert, und einer Tabelle, in der die benötigten bedingten Wahrscheinlichkeiten gespeichert werden. Das Trainieren von Bayes'schen Netzen kann sich sowohl auf das Erlernen der Abhängigkeiten zwischen den einzelnen Attributen, also auf die Struktur des Netzes, als auch auf das Erlernen der Tabelleneinträge, also auf die notwendigen bedingten Wahrscheinlichkeiten, beziehen. Für das Erlernen der Tabelleneinträge wurde ein Algorithmus vorgestellt, der einen Gradientenabstieg durchführt.

Die intuitive Behandlung der Konzepte der *Regression* zur Lösung des Prädiktionsproblems war Inhalt von Kapitel 5. Die *lineare Regression* modelliert eine Antwortvariable Y durch eine lineare Funktion, die auf eine Schätzervariable X angewendet wird. Die *multiple Regression* erlaubt dagegen statt einer einzigen Schätzervariablen einen mehrdimensionalen Attributvektor. Bei der linearen und bei der multiplen Regression wird üblicherweise die *Methode der kleinsten Quadrate* zur Ermittlung der *Regressionskoeffizienten* angewendet. Bei der *polynomialen Regression* wurde ein nichtlineares Modell durch Substitution von polynomialen Variablen in ein lineares Modell überführt, das nun auch durch die Methode der kleinsten Quadrate lösbar ist.

Abschließend bleibt zu sagen, dass die verhältnismäßig neue Disziplin des Data Mining althergebrachte Klassifikationskonzepte der Statistik und der Künstlichen Intelligenz nutzt und ständig weiterentwickelt. Aufgrund des rasanten Wachstums der zu bearbeitenden Datenmengen wird der Bedarf an immer schnelleren Algorithmen in absehbarer Zeit nicht abreißen, wodurch die Klassifikation ein interessantes Forschungsgebiet bleibt.

Literatur

[AIS93] R. Agrawal, T. Imielinski and A. Swami. *Data Mining: A performance perspective*. IEEE Transactions on Knowledge and Data Engineering, 5(6):914-925, December 1993.

[AMR96] R. Agrawal, M. Mehta and J. Rissanen. *SLIQ: A fast scalable classifier for data mining*. In Proc. Of the 5th Int'l Conference on Extending Database Technology (EDBT), Avignon, France, 1996.

[AMS96] R. Agrawal, M. Mehta and J. Shafer. *SPRINT: A Scalable Parallel Classifier for Data Mining*. In Proc. of the 22nd VLDB Conference, Bombay, India, 1996.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[HK01] J. Han and M. Kamber. *Data Mining - Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[MST94] D. Michie, D.J. Spiegelhalter and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[NASA92] NASA Ames Research Center. *Introduction to IND Version 2.1*. GA23-2475-02 edition, 1992.

[Qui86] J.R. Quinlan. *Induction of decision trees*. Machine Learning(1), 1986.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1993.

[Ri89] J. Rissanen. Stochastic Complexity in Statistical Inquiry. World Scientific Publ. Co., 1989.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1995.