Zusammenfassung

Die vorliegende Arbeit befaßt sich mit der Aggregation von Attributwerten einer relationalen Datenbank. Hierzu werden zwei Verfahren vorgestellt: CLUST_DB gruppiert Werte nominaler Attribute gemäß einer Cluster-Analyse unter Verwendung eines eigens für das Verfahren definierten Ähnlichkeitsmaßes. NUM_INT sucht in numerischen Attributen nach Intervallen, die sich als sinnvolle Vergröberung der einzelnen Werte anbieten und stellt damit ein Diskretisierungsverfahren dar.

Beide Verfahren arbeiten auf einer relationalen Datenbank und können als Werkzeug bei der Wissensentdeckung in Datenbanken (KDD) verwendet werden: CLUST_DB kann der Entdeckung und Überprüfung einstelliger funktionaler Abhängigkeiten dienen und für Begriffslerner eine effektive Dimensionsreduktion bei klassifizierten Beispielen erzielen. Darüber hinaus kann es wie auch NUM_INT zur Überprüfung der Datenqualität genutzt werden.

Die zugrundeliegenden Ansätze sind als solche nicht an eine relationale Datenbank gebunden, so daß beide Verfahren auch dann anwendbar bleiben, wenn Daten in anderer Form vorliegen.

Danksagung

Ich möchte mich an dieser Stelle bei all denen bedanken, die in besonderer Weise zur Fertigstellung dieser Diplomarbeit beigetragen haben:

Prof. Dr. Katharina Morik möchte ich sehr dafür danken, daß sie mich durch ihre Lehre mit dem Gebiet des maschinellen Lernens und des KDD bekannt gemacht hat und mir durch ihre Betreuung die Bearbeitung der Aufgabenstellung ermöglichte. Siegfried Bell bin ich sehr dankbar, daß er trotz abschließender Arbeiten an seiner Dissertation stets Zeit für Diskussionen und Verbesserungsvorschläge fand und wünsche ihm an dieser Stelle alles Gute für das anstehende Promotionsverfahren.

Neben den beiden genannten fand ich in Peter Brockhausen einen weiteren Partner, der mich u. a. mit der Ankopplung und Handhabung von relationalen Datenbanken vertraut machte. Bei Michael Goebel möchte ich mich für die Diskussion über die Cluster-Analyse bedanken, die unsere beiden Diplomarbeiten verbindet; bei Andreas Greve dafür, daß die Rechner und Drucker stets einsatzbereit waren und er rasch zur Stelle war, wenn es Probleme gab. Meinen Kollegen Thorsten Kitz, Daniel Boettcher und Dietmar Senkbeil möchte ich für deren Unterstützung im allgemeinen und den Crash-Kurs *awk* sowie Beantwortung meiner Fragen zu C im besonderen danken.

Neben ML, KDD, KRK, SQL und kNN gibt es glücklicherweise auch noch andere Dinge im Leben, ohne die es sich aber letztlich schlecht studieren ließe. So möchte ich meinem Freund Heiner Post dafür danken, daß er mit sprachlichem Geschick und der nötigen Übersicht half, der Diplomarbeit den letzten Schliff zu geben. Ganz besonders herzlich möchte ich meinen Eltern Horst und Ursula, meiner Schwester Silke und meiner Frau Yvonne dafür danken, daß sie stets für mich da waren und hoffe, daß ich nun wieder mehr Zeit für sie und unseren kleinen Jonas haben werde.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die 90er Jahre stehen unbestritten im Zeichen des rapiden Fortschritts auf dem Gebiet der Informationstechnik, der zu einer wahren Flut von Daten führte. Mit dieser Entwicklung entstand das zunehmende Bedürfnis, diese Daten zu speichern, ihnen möglichst viel *Information* zu entnehmen. In diesem Sinne hat sich das Volumen der Datenbanken in den letzten Jahren vervielfacht, wodurch die Anforderungen an Verfahren, die diese analysieren sollten, immens stiegen. Doch wie [FPSSU96, Vorwort] bemerken, hielten die vorhandenen Möglichkeiten, Daten zu analysieren, zusammenzufassen und ihnen Wissen abzugewinnen, nicht mit der technischen Entwicklung Schritt, die die Anhäufung immer größerer Datenmengen ermöglichte.

Aus dieser Situation heraus entstand das Gebiet der Wissensentdeckung in Datenbanken (*engl.: Knowledge Discovery in Databases (KDD)* oder auch *Data Mining*) [PSF91], das seinen besonderen Reiz neben dem großen praktischen Nutzen daraus bezieht, daß es hier gilt, Ideen und Konzepte u. a. der Statistik, der Datenbanktheorie und des maschinellen Lernens zu bündeln.

Den wichtigsten Beitrag seitens des maschinellen Lernens stellt die Entwicklung von Verfahren dar, die aus einer gegebenen Menge an Daten *lernen*[1]. Für diese Lernverfahren gilt es im Kontext des KDD, ihren Dienst auf den Daten einer Datenbank zu verrichten, was nicht nur heißt, daß sie mit sehr viel mehr Daten konfrontiert werden, als bisher üblich[2], sondern daß auch der Zugriff auf die Daten den Gegebenheiten der Datenbank angepaßt werden muß: Die Datenbanken sind typischerweise so groß, daß Anfragen an die Datenbank und Zugriffe auf die Daten selbst nur über eine Schnittstelle wie SQL sinnvoll erfolgen können — an ein Kopieren in den Hauptspeicher mit daraus resultierendem beliebigen Zugriff ist nicht zu denken.

Schenkten bis vor einiger Zeit Lernverfahren diesem Aspekt kaum Aufmerksamkeit, so ist es spätestens jetzt von größtem Interesse, Verfahren zu konzi-

[1] Eine allgemeine Einführung in das Gebiet des (maschinellen) Lernens findet sich in [Mor93].

[2] Eine klassische Lernaufgabe stellt das spezielle Schachendspiel *KRK* [Qui83] dar, bei dem sich nur noch die beiden Könige (*engl. kings*) und ein weißer Turm (*engl. rook*) gegenüberstehen und zu lernen ist, welche konstruierten Stellungen den Regeln entsprechen und welche nicht. Diese Domäne umfaßt im Original 10.000 Stellungen, während Datenbanken mit mehr als 250.000 Einträgen gewiß keine Besonderheit darstellen.

pieren, die effizient auf realen Datenbanken arbeiten können. Einen wichtigen Schritt in diese Richtung stellt das Lernverfahren RDT/DB [Lin94] dar, welches als Adaption des *Rule Discovery Tool* (RDT) [KW92] direkt an eine relationale Datenbank gekoppelt nach Regelhaftigkeiten sucht.

Diese Diplomarbeit wird sich mit der Aufgabe befassen, innerhalb der einzelnen Attributwerte einer relationalen Datenbank Gruppierungen zu entdecken, die für das Lernen genutzt werden können. Diese Aufgabe kann aus unterschiedlichen Blickwinkeln motiviert werden: Zum einen sollen die Gruppierungen zu einer gewissen Datenreduktion führen — bei möglichst geringem Informationsverlust. Man spricht dabei im allgemeinen von *Aggregation*, bei (kontinuierlichen) numerischen Daten von *Diskretisierung* bzw. *Kategorisierung*. Den letzteren kommt besondere Bedeutung zu, da viele Lernverfahren nicht oder nur schlecht mit numerischen Werten arbeiten können.

Zum anderen besitzt dieser Gruppierungsschritt auch begriffsbildenden Charakter: Werden die Gruppen benannt, so stellen diese neue Begriffe dar. Können diese zur Erhöhung der Ausdruckskraft genutzt werden, spricht man in diesem Zusammenhang von *predicate invention*.

Der folgenden Abschnitt dient dazu, die Aufgabenstellung zu konkretisieren.

1.1 Aufgabenstellung

In der Diplomarbeit werden zwei Aufgaben im Gebiet der Wissensentdeckung in Datenbanken (KDD) bearbeitet. Beiden gemein ist die Zielsetzung, Lernverfahren den Boden zu bereiten: Es sollen anhand der Attributwerte einer relationalen Datenbank Sorten induziert und verfügbar gemacht werden, die es etwa RDT(/DB) [KW92] [Lin94] ermöglichen, den Hypothesenraum zu beschneiden.

Dieser Aufgabe kommt beim KDD besondere Bedeutung zu, da der Test einer Hypothese bei einer Datenbank sehr zeitaufwendig und damit teuer ist. Die Datenbanken sind typischerweise so groß, daß Anfragen an die Datenbank und Zugriffe auf die Tupel selbst nur über eine Schnittstelle wie SQL sinnvoll erfolgen können.

Daß es sich bei der Diplomarbeit um die Bearbeitung zweier Aufgaben handelt, ergibt sich aus der getrennten Behandlung von numerischen und nominalen (symbolischen) Attributen: Bei den erstgenannten sollen Intervalle gefunden werden, die eine sinnvolle (hierarchische) Vergröberung der einzelnen numerischen Daten darstellen. Dabei ist insbesondere zu prüfen, ob die Adaption bereits bestehender Verfahren auf eine Datenbank vorteilhaft ist, oder ob ein neuer Algorithmus zu entwickeln ist, der bessere Ergebnisse liefert — möglicherweise basierend auf dem in [Wes95] beschriebenen Ansatz.

Bei den nominalen Attributen sollen Mengen von Attributwerten gefunden werden, die als Sorten zusammengefaßt werden können. Es ist dabei insbesondere von Interesse, ob die durch das Verfahren FDD [Bro94] in der Datenbank gefundenen funktionalen Abhängigkeiten dafür auszunutzen sind.

1.2 Übersicht

Die Arbeit gliedert sich in fünf Teile, denen jeweils ein Kapitel gewidmet ist. Das erste Kapitel — die Einleitung — endet mit dieser Übersicht.

Im zweiten Kapitel werden zunächst die für das Verständnis der Arbeit nötigen Voraussetzungen in Bezug auf die Datenbanktheorie geschaffen. Dies soll auf möglichst anschaulichem Weg geschehen, um insbesondere die Einladung an diejenigen Leser zu bekräftigen, die mit dieser Materie nicht vertraut sind.

Es folgt das dritte Kapitel, in dem das Verfahren CLUST_DB vorgestellt wird, welches vorrangig für die Behandlung der nominalen Attribute entwickelt wurde.

Das vierte Kapitel befaßt sich mit den numerischen Attributwerten. In ihm wird das Verfahren NUM_INT vorgestellt, diskutiert und analysiert.

Das fünfte Kapitel dieser Arbeit trägt die Ergebnisse der Kapitel 3 und 4 für eine abschließende Betrachtung zusammen, bei der auch ein Ausblick auf Zukünftiges erfolgen soll.

Kapitel 2

Relationale Datenbanken

In diesem Kapitel wird das relationale Datenbankmodell erläutert. Für das Verständnis der Arbeit genügt es, dieses recht informell einzuführen, so daß auf die Darstellung des zugrundeliegenden mathematischen Modells der Relationen-Algebra weitestgehend verzichtet werden kann. Es wird vielmehr für die Beschreibung der anschauliche Ansatz über Tabellen gewählt und lediglich die Projektion π und die Selektion σ kurz erklärt.[1]

Datenbank, Attribute und Tupel

Eine relationale Datenbank besteht aus einer endlichen Menge unterschiedlicher Tabellen, die auch als *Relationen* bezeichnet werden. Jede dieser Tabellen hat einen eindeutigen Namen und besteht aus Spalten und Zeilen, die mit endlich vielen Einträgen gefüllt sind. Die Spalten werden *Attribute*, die Zeilen *Tupel* genannt. Letztere bestehen aus je einem Wert pro Attribut. Ein besonderer Wert ist die $NULL$, die anzeigt, daß kein Eintrag vorliegt. Im Gegensatz dazu stellen eine aus Leerzeichen bestehende Zeichenkette oder die Zahl 0 sehr wohl Einträge dar.

Die Bezeichnung eines Attributs muß insofern eindeutig sein, als daß sie in jeder Tabelle höchstens einmal vorkommt, so daß über die Angabe der Attributbezeichnung und des Tabellennamens genau eine Spalte identifiziert werden kann. Eine solche Tabelle einer Datenbank wird in Tabelle 2.1 beispielhaft gezeigt: Sie trägt die Bezeichnung $LIEFERANT$ und besteht aus 5 Attributen und 10 Tupeln.

Sowohl die Reihenfolge der Attribute als auch die der Tupel ist an sich wegen der Mengeneigenschaft beliebig, kann für letztere aber durch eine Sortierung anhand eines oder mehrerer Attribute erzwungen werden. Für jedes Attribut muß ein Typ angegeben werden, der festlegt, welche Art von Werten in der entsprechenden Spalte eingetragen werden darf. Im Rahmen dieser Arbeit wird zwischen *nominalen, numerischen* und *binären*[2] Attributen unterschieden:

[1] Interessierte Leser seien bzgl. einer ausführlichen Darstellung der Relationen-Algebra etwa auf [Ull88, Kap. 3] verwiesen.

[2] Die binären Attribute spielen für die Aufgabenstellung keine Rolle, da es bei nur zwei möglichen Werten keinen Sinn macht, nach Gruppierungen zu suchen. Sie kommen lediglich im Abschnitt 3.1.2 über Maße der Cluster-Analyse vor.

LIEFERANT	Name	Standort	Ware	Preis	Datum
	Farmer-Joe	DO	Geflügel	105.10	01.07.96
	Farmer-Joe	DO	Gemüse	30.30	01.07.96
	Bauer-Bill	OB	Kartoffeln	76.00	03.07.96
	Mc-Dorado	E	Geflügel	18.15	04.07.96
	Mc-Dorado	E	Mais	34.21	01.07.96
	Mc-Dorado	E	Fisch	85.40	04.07.96
	Hof-Holt	MH	Obst	20.00	15.07.96
	Mc-Dorado	E	Fisch	74.50	11.07.96
	Hof-Holt	MH	Gemüse	32.60	18.07.96
	Fischers-Fritze	E	Fisch	83.65	10.08.96

Tabelle 2.1: Beispiel einer möglichen Datenbanktabelle eines Restaurants

nominal: Die Attributwerte besitzen keine (sinntragende) Ordnung. Es handelt sich bei den Attributwerten typischerweise um Kodierungen oder Bezeichner. Attribute diesen Typs werden oft auch *symbolisch* oder *kategorisch* genannt. Beispiele hierfür sind die Attribute $Name$, $Standort$ und $Ware$ in Tabelle 2.1.

numerisch: Die Werte stammen aus einem Ausschnitt der natürlichen oder reellen Zahlen. In Tabelle 2.1 kommt das numerische Attribut $Preis$ vor.[3]

binär: Ein binäres Attribut verfügt nur über die zwei möglichen Werte 0 und 1, die dann als *wahr / falsch*, *vorhanden / nicht vorhanden* etc. interpretiert werden können. In Tabelle 2.1 kommt kein binäres Attribut vor.

Dem Zugriff auf die Daten einer Tabelle dienen die 5 Grundoperationen der relationalen Algebra: Vereinigung, Mengendifferenz, kartesisches Produkt, Projektion und Selektion. In der Arbeit werden lediglich die beiden letztgenannten verwendet. Sie werden im folgenden erläutert:

Während R die gesamte Relation über n Attribute, also alle n-Tupel der Tabelle beinhaltet, können mittels *Projektion* π die Tupel auf angegebene Attribute beschränkt werden. So gilt für die in Tabelle 2.1 dargestellte Relation $LIEFERANT$:

$$\begin{aligned}\pi_{Standort,Ware}(LIEFERANT) \quad = \quad & \{(DO, Geflügel), (DO, Gemüse), \\ & (OB, Kartoffeln), (E, Geflügel), \\ & (E, Mais), (E, Fisch), \\ & (MH, Obst), (MH, Gemüse)\}\end{aligned}$$

[3] Das Attribut *Datum* ist in diesem Sinne nicht numerisch, kann aber von dem vorzustellenden Verfahren NUM_INT als solches behandelt werden, da eine sinnvolle Ordnung existiert und eine Differenzbildung möglich ist.

Die *Selektion* wählt mittels σ_φ ausschließlich die Tupel der Relation aus, deren Werte der angegebenen logischen Bedingung φ genügen. So gilt etwa

$$\begin{aligned}\sigma_{Standort=DO}(LIEFERANT) &= \{(Farmer-Joe, DO, Geflügel, 105.10, 01.07.96),\\ &\quad (Farmer-Joe, DO, Gemüse, 30.30, 01.07.96)\}\end{aligned}$$

Beide Operationen können auch kombiniert werden, wie folgendes Beispiel zeigt:

$$\pi_{Standort,Ware}(\sigma_{Standort=DO}(LIEFERANT)) = \{(DO, Geflügel), (DO, Gemüse)\}$$

Es ist im relationalen Modell jedoch nicht vorgesehen, auf die Attributwerte eines Tupels durch Angabe einer Zeilennummer zuzugreifen.

Relationale Datenbanksysteme

Ein relationales Datenbanksystem besteht aus der Datenbank (DB) und einem Datenbankmanagementsystem (DBMS), welches u. a. dem Benutzer einen möglichst komfortablen und effizienten Zugriff auf die physikalisch in der Datenbank gespeicherten Daten gestattet. In der Regel wird als Anfragesprache SQL benutzt, die in diesem Kontext als Umsetzung der zuvor beschriebenen Operationen der relationalen Algebra betrachtet werden kann.[4] Auf sie wird am Ende dieses Kapitels eingegangen. Darüber hinaus wacht das DBMS auch über die Einhaltung des Typs, der jedem Attribut beim Aufstellen der Tabelle zugewiesen wurde.

SQL

Die *Standard Query Language* (SQL) löst ein, was der Name bereits verspricht: Sie gilt als Standard-Anfragesprache für relationale Datenbanken.[5] Mit ihrer Hilfe werden auch die Datenbankzugriffe der vorzustellenden Verfahren formuliert, weshalb die benutzten Konstrukte kurz erklärt und an einem Beispiel illustriert werden sollen. Die Sprache selbst ist bedeutend mächtiger, wovon man sich etwa in [Dat95] leicht überzeugen kann.

1. `select distinct ( A ) from R ordered by A` übergibt die sortierte Menge der Attributwerte von A. Der Mengencharakter wird durch das Schlüsselwort `distinct` sichergestellt, das doppelte Werte herausfiltert. Abgesehen von der Ordnung bzgl. A entspricht diese Anfrage somit der Projektion $\pi_A(R)$.

2. `select distinct ( A, B ) from R ordered by A` resultiert analog zu obigem in einer Menge von Paaren (a, b), wobei a ein Attributwert von A und b ein Attributwert von B ist. Das `order by A` bewirkt, daß das Ergebnis bzgl. A sortiert wird.

[4] Bei genauerer Betrachtung der Materie läßt sich feststellen, daß es sich keineswegs um eine saubere 1:1-Umsetzung handelt, was aber für die Arbeit keine Rolle spielt und daher hier nicht weiter diskutiert werden soll.

[5] Im vorliegenden Fall wurde das SQL2 des DBMS Oracle V7 der Firma Oracle benutzt.

Beispiel 2.1 *Für $Lieferung$ aus Tabelle 2.1 erhält man die folgenden Resultate:*

1. `select distinct (` $Ware$ `) from` $LIEFERANT$ `ordered by` $Ware$
   $\rightarrow \{Fisch, Geflügel, Gemüse, Kartoffeln, Mais, Obst, Rind\}$

2. `select distinct (` $Standort, Datum$ `) from` $LIEFERANT$
   `ordered by` $Standort$
   $\rightarrow \{(DO, 01.07.96), (E, 04.07.96), (E, 01.07.96),$
   $(OB, 03.07.96), (OB, 18.07.96), (MH, 15.07.96)\}$

Funktionale Abhängigkeiten

Im Verlauf der Arbeit wird das Konzept der *funktionalen Abhängigkeiten (FA)* genutzt. Es sei daher an dieser Stelle definiert und an einem Beispiel illustriert.

Definition 2.1 *Seien X, Y zwei Mengen von Attributen einer Tabelle einer relationalen Datenbank. Dann gilt die* **funktionale Abhängigkeit (FA)** *$X \rightarrow Y$ genau dann, wenn die Attribute aus X keine Nullwerte enthalten und alle Tupel, die in X übereinstimmen, auch in Y übereinstimmen.*

Beispiel 2.2 *Für Tabelle 2.1 gilt etwa in trivialer Weise die FA $\{Preis, Datum\} \rightarrow \{Name, Ware, Standort\}$, da es keine zwei Tupel gibt, die in $Preis, Standort$ übereinstimmen. Des weiteren gilt $Name \rightarrow Standort$[6], da in der angegebenen Tabelle jeder Name einem Standort entspricht. Es gilt nicht $Standort \rightarrow Name$, da es zwei Einträge gibt, die den gleichen Standort, aber unterschiedliche Namen aufweisen.*

[6]Man beachte, daß die Mengenklammer weggelassen wird, wenn die Attributmengen X, Y einelementig sind. Man spricht dann auch von *einstelligen* FA.

Kapitel 3

Aggregation in nominalen Attributen

In diesem Teil der Arbeit wird die Behandlung der symbolischen oder auch *nominalen* Attribute im Vordergrund stehen. Es wird gemäß der Aufgabenstellung das Ziel verfolgt werden, in diesen Wertebereiche zu finden, die einem Lernverfahren als Typ nützen können.

Im Gegensatz zur später folgenden Behandlung der numerischen Attribute kann weder direkt auf eine Klassifikation noch auf eine innere Struktur der Attributwerte zurückgegriffen werden — auf die Ausnutzung der selbstverständlich möglichen lexikographischen Ordnung sollte sinnvollerweise verzichtet werden. Es liegt somit eine Ausgangssituation vor, die eine Herangehensweise gemäß der Cluster-Analyse nahelegt. Dies sollte Motivation genug sein, einen Abschnitt folgen zu lassen, der als Einführung in die Cluster-Analyse dienen soll.

Anschließen wird die Darstellung eines Cluster-Ansatzes zur Lösung des vorliegenden Problems und die Beschreibung der einzelnen Komponenten des im Rahmen dieser Diplomarbeit entwickelten CLUST_DB. Es folgen Abschnitte über die Implementierung und Versuchsergebnisse, ehe CLUST_DB mit anderen Verfahren verglichen wird und das dritte Kapitel mit einer Diskussion der erzielten Ergebnisse schließt.

3.1 Cluster-Analyse

Es wird in diesem Abschnitt darum gehen, die wesentlichen Merkmale unterschiedlicher Cluster-Analyse-Ansätze zu skizzieren. Die Betrachtung dieser Klasse von Algorithmen ist motiviert durch die eingangs beschriebene Problemstellung, die keinen Rückgriff auf eine natürliche Klassifikation zuläßt, sondern ein Lernen aus unklassifizierten *Beobachtungen* erfordert.

Die Cluster-Analyse kann auf eine sehr lange Entwicklungsgeschichte zurückblicken. Auch heute noch stellt sie ein vieldiskutiertes Objekt der Wissenschaft dar, wie [Mur92] eindrucksvoll aufzeigt. So findet sie im Bereich der künstlichen Intelligenz ebenso Verwendung wie etwa bei der Datenanalyse, Linguistik, Medizin und in den Sozialwissenschaften.

Dies erklärt auch, warum dieser Abschnitt sich auf die wesentlichen Merkmale bekannterer Ansätze beschränken muß. Eine über die folgende Einführung hinausgehende Schilderung läßt sich u. a. in [Eve80], [Gna88], [Hub92] und [MHMT92] finden.

Das durch die Cluster-Analyse zu lösende Problem stellt sich wie folgt dar:

Gegeben: Menge von Objekten — beschrieben durch Attributwerte

Gesucht: Klasseneinteilung der Objekte, so daß Objekte der gleichen Klasse *ähnlich* und Objekte verschiedener Klassen *unähnlich* sind.

Wie hieraus bereits hervorgeht, ist die Frage nach der Definition der Ähnlichkeit bei der Cluster-Analyse von entscheidender Bedeutung. Im Abschnitt 3.1.2 wird darauf genauer eingegangen, indem unterschiedliche Ähnlichkeits- bzw. Distanzmaße vorgestellt werden.

In der obigen Problembeschreibung wird angedeutet, daß in den allermeisten Cluster-Verfahren unter einem Objekt ein Tupel von Attributwerten verstanden wird. Es sollte festgehalten werden, daß es im Gegensatz dazu im vorliegenden Fall darum geht, Attributwerte eines ausgezeichneten Attributes — also eine Komponente eines Tupels — zu gruppieren. Es wird sich zeigen, daß diese andere Sichtweise entsprechende Überlegungen bezüglich der Wahl des Ähnlichkeitsmaßes erfordert: Die Ähnlichkeit unter den einzelnen Attributwerten ist bei der Cluster-Analyse bei den nominalen Attributen oft entweder trivial (etwa 0 für Ungleichheit, 1 für Gleichheit) oder vom Benutzer vorzugeben. Dies wird im Abschnitt 3.1.2 näher betrachtet werden.

Gemein ist den im Abschnitt 3.1.2 vorzustellenden Maßen, daß sie für eine Überführung der gegebenen Objektbeschreibungen in eine Matrix sorgen, die die Ähnlichkeit bzw. die Distanz zwischen den einzelnen Objekten wiedergibt. Diese Matrix wird genutzt, um eine Gruppierung — oft auch als Klasseneinteilung bezeichnet — zu induzieren. Dies kann durchaus auch als Versuch betrachtet werden, eine Vereinfachung zu finden, die maximale Datenreduktion bei minimalem Informationsverlust bietet [Eve80, S. 6].

Die Art und Weise, in der aus der Matrix die Klassen (im folgenden auch als Gruppierungen oder Gruppen bezeichnet) gewonnen werden, wird Gegenstand des Abschnitts 3.1.3 sein. Darin werden auch Ansätze vorgestellt, die die Güte eines Clusters nicht (ausschließlich) aus der Ähnlichkeits- bzw. Distanzmatrix ableiten, sondern alternative Bewertungen für den Gruppierungsschritt hinzuziehen.

Die Attribute der Objektbeschreibungen verdienen im allgemeinen besondere Aufmerksamkeit beim Einsatz eines Cluster-Verfahrens — und das in mehrfacher Hinsicht: Zum einen muß bei der Datenerfassung bzw. -selektion entschieden werden, welche Attribute als Objektbeschreibungen dienen sollen. Nicht selten steigt der Aufwand der Cluster-Analyse stark mit der Anzahl der betrachteten Attribute, so daß ggf. zuvor erfaßte Attribute bei der Analyse aus praktischen Gründen vernachlässigt werden müssen. Es stellt sich dabei ganz natürlich die Frage, auf welche am ehesten verzichtet werden kann. Diese Thematik wird in der Literatur auch häufig unter dem Begriff *feature selection* diskutiert.

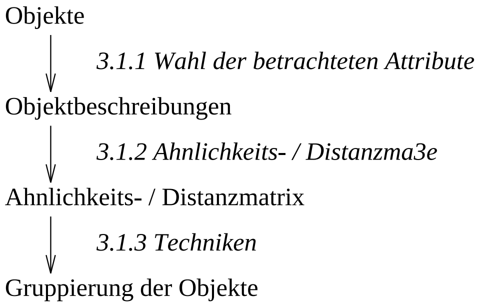

Abbildung 3.1: Schritte der Cluster-Analyse

Zum anderen gesellt sich dazu das Problem der Gewichtung der zu betrachtenden Attribute: Sollen alle gewählten Attribute gleichberechtigt über die Klasseneinteilung der Objekte entscheiden oder soll einigen dabei größerer Einfluß als anderen eingeräumt werden?

Während das erstgenannte Problem bzgl. der Wahl der Attribute bei der Datenerfassung hier nicht auftritt — die Daten liegen bereits in der Datenbank gespeichert vor — bedürfen die anderen Punkte der näheren Betrachtung: Abschnitt 3.1.1 wird sich damit befassen.

Es sollte bei dem in Abbildung 3.1 skizzierten modularem Aufbau bei recht unabhängiger Wahl der Einzelteile eines Cluster-Analyse-Verfahrens nicht verwundern, daß es eine Vielzahl von unterschiedlichen Ansätzen gibt. Es wird in den folgenden Abschnitten darum gehen, die am häufigsten verwendeten bzw. die im Hinblick auf die Aufgabenstellung interessantesten Ansätze zu umreißen.

3.1.1 Wahl und Gewichtung der Attribute

Es gibt gute Gründe, bei dem Einsatz eines Cluster-Verfahrens sorgfältig zu überlegen, welche der erfaßten Attribute in die Analyse eingehen: Für die Gruppierung irrelevante Attribute können vorhandene „sinnvolle" Gruppen verwässern oder gar verschütten — abgesehen davon, daß die Analyse durch die unnötig hohe Attributanzahl länger dauert. Doch auch wenn alle (verbliebenen) Attribute für die Aufteilung in Gruppen von Bedeutung sind, lohnt es sich bzw. ist es erforderlich, einen genaueren Blick auf diese zu werfen. So können korrelierte Attribute die Analyse in unerwünschter Weise beeinflussen: Betrachtet man etwa als Beispiel den einfachsten Fall einer Korrelation, in dem ein Attribut doppelt (unter einer anderen Bezeichnung) erfaßt wurde, so wird klar, daß dieses Attribut quasi zweimal seine Stimme bei der Berechnung der Ähnlichkeit / Distanz und damit letztendlich für die Wahl der Gruppierung abgibt.

Des weiteren wird es in aller Regel so sein, daß die relevanten Attribute nicht alle den gleichen Stellenwert bzgl. der Gruppierung aufweisen (sollen): Geht es etwa darum, per Cluster-Analyse Fortbewegungsmittel zu gruppieren, so sollte dem Attribut *Antriebsart* größeres Gewicht eingeräumt werden als dem Attribut *Preis*.

Neben dem (hoffentlich vorhandenen) Wissen über die Domäne, das bei all diesen Punkten auf natürliche Weise in die Cluster-Analyse einfließen kann, kann die *Principal Component Analysis (PCA)* herangezogen werden. Sie bestimmt gemäß der in der Informationstheorie als Karhunen-Loéve-Expansion bekannten Transformation die eigenständigen, d. h. paarweise unkorrelierten Komponenten und gibt mit Hilfe von Matrix-Eigenwerten Auskunft über deren Varianz. Sie kann neben der Beseitigung der Korrelationen auch bei den anderen beiden Punkten sinnvoll eingesetzt werden:

Datenreduktion: Es werden nur die Attribute mit den p' größten Eigenwerten berücksichtigt.

Gewichtung: Die Attribute werden durch ihre bei der PCA bestimmten Eigenwerte gewichtet.

Obwohl theoretisch das Attribut mit der kleinsten Varianz am aussagekräftigsten sein kann, bestätigt Wettschereck in seiner Studie über *Distance-Based Machine Learning Algorithms* [Wet94] die Tauglichkeit des obigen Ansatzes weitestgehend. Insbesondere die Datenreduktion durch PCA erwies sich als erfolgreich: So konnte man sich in einem Fall[1] gar auf die 2 Attribute mit den größten Eigenwerten beschränken, ohne an Vorhersagekraft zu verlieren. An gleicher Stelle wird aber auch festgehalten, daß es bislang nicht gelang, anhand der Eigenwerte einen Schwellenwert zu bestimmen, bis zu welchem die Attribute betrachtet werden sollten.

Wenn man über den Einsatz der PCA spricht, so müssen allerdings sicher auch die beiden folgenden Nachteile genannt werden:

- Die Rechenintensität steigt stark mit der Anzahl der Dimensionen (Attribute) an.
- Die transformierten Attribute haben keinen offensichtlichen Bezug zu den ursprünglichen und entziehen sich dadurch einer sinnvollen Interpretation.

In Abschnitt 3.2.6 wird die Anwendbarkeit der PCA im vorliegenden Fall diskutiert.

3.1.2 Ähnlichkeits- und Distanzmaße

Den Kern eines Cluster-Verfahrens bildet das verwendete Maß, das angibt, wie ähnlich sich zwei Objekte sind (Ähnlichkeitsmaß) bzw. wie weit zwei Objekte von einander entfernt sind (Distanzmaß). Es ist unbestritten, daß das Ergebnis eines Cluster-Verfahrens nur so sinnvoll sein kann wie das zugrundeliegende Maß.

Bevor im einzelnen unterschiedliche Konzepte für Ähnlichkeiten und Distanzen hier vorgestellt werden, soll zunächst die Beziehung Ähnlichkeit — Distanz geklärt werden.

[1] Es handelt sich dabei um die künstliche Domäne *Waveform-21* [BFOS84] mit 21 numerischen Attributen.

Definition 3.1 *Eine numerische Funktion $d : E \times E \rightarrow \mathbb{R}$ auf einer Menge E von Punkten wird genau dann* **Distanzmaß (Metrik)** *genannt, wenn sie die folgenden Voraussetzungen erfüllt:*

(D1) $d(x,y) \geq 0$; $d(x,y) = 0 \Leftrightarrow x = y$ *(positive Definitheit)*

(D2) $d(x,y) = d(y,x)$ *(Symmetrie)*

(D3) $d(x,y) + d(y,z) \geq d(x,z)$ *(Dreiecksungleichung)*

Im Gegensatz zum Distanzmaß lassen sich in der Literatur unterschiedliche Auffassungen darüber finden, welchen Bedingungen ein Ähnlichkeitsmaß formal zu genügen hat. Eine in diesem Sinne „strenge" Auslegung — siehe Fußnoten — stellt Definition 3.2 dar:

Definition 3.2 *Eine numerische Funktion $s : E \times E \rightarrow [0;1]$[2] auf einer Menge E von Objekten wird genau dann* **Ähnlichkeitsmaß** *genannt, wenn sie die folgenden Voraussetzungen erfüllt:*

(S1) $s(x,x) = s(y,y) = 1 \geq s(x,y)$ *für alle* $x, y \in E$ *(Reflexivität)*

(S2) $d(x,y) = d(y,x)$ *(Symmetrie)*[3]

Aus einem Distanzmaß läßt sich leicht etwa durch $s(x,y) := 1/(1 + d(x,y))$ ein Ähnlichkeitsmaß gewinnen. Die andere Richtung ist durchführbar aber keineswegs trivial, was ob der offensichtlich schärferen Forderungen an ein Distanzmaß nicht verwundert — insbesondere die Dreiecksungleichung (D3) gilt es zu bewahren. [Gow66] zeigte beispielsweise, daß $d(x,y) := \sqrt{2(1 - s(x,y))}$ als Distanzmaß dienen kann, sofern die Ähnlichkeitsmatrix S positiv semi-definit ist.

Ähnlichkeit, Unähnlichkeit, Distanz werden in der Literatur (siehe etwa [PA87]) häufig zusammen mit anderen als *proximities* bezeichnet, da viele Gruppierungstechniken diesbezüglich flexibel sind.

Ein Distanz- bzw. Ähnlichkeitsmaß muß in drei unterschiedlichen Situationen greifen: Objekt–Objekt, Objekt–Gruppe und Gruppe–Gruppe. Es werden zunächst Objekt–Objekt- und dann Gruppe–Gruppe-Maße vorgestellt. Die Situation Objekt–Gruppe wird nicht gesondert betrachtet, da sie in den allermeisten Fällen durch Betrachtung des Objekts als einelementige Gruppe auf den Fall Gruppe–Gruppe reduziert wird.

Objektähnlichkeit

Sind die Werte der betrachteten Attribute vom Typ *vorhanden / nicht vorhanden*, also **binär**, dann bietet es sich für zwei zu bemessende Objekte an, das gemeinsame Vorkommen bzw. Fehlen von Werten anhand der in Abbildung 3.2 dargestellten Assoziationstabelle zu ermitteln, und daraus einen Ähnlichkeitswert zu bestimmen: a gibt beispielsweise an, in wievielen Attributen die beiden Objekte X, Y übereinstimmend ein *vorhanden* bzw. 1 aufweisen.

[2] Es läßt sich in der Literatur auch $s : E \times E \rightarrow \mathbb{R}^{+}$ finden.

[3] Die Symmetrie wird oft angenommen, aber seltener explizit gefordert.

		Objekt j	
		1	0
Objekt i	1	a	b
	0	c	d

Abbildung 3.2: Binäre Assoziationstabelle für zwei Objekte

(i)	$\frac{a+d}{p}$	(ii)	$\frac{a}{a+b+c}$	(iii)	$\frac{2a}{2a+b+c}$
(iv)	$\frac{2(a+d)}{2(a+d)+b+c}$	(v)	$\frac{a}{a+2(b+c)}$	(vi)	$\frac{a}{p}$

Tabelle 3.1: Ähnlichkeitskoeffizienten für binäre Daten

Ist p die Anzahl der gemeinsamen Attribute, so gilt offensichtlich $p = a + b + c + d$. In der Literatur wird darauf aufbauend eine Vielzahl von unterschiedlichen Formeln zur Berechnung der Ähnlichkeit von X, Y vorgeschlagen, welche insbesondere durch die Unsicherheit erklärt wird, wie der Wert d zu handhaben ist: In einigen Fällen mag es angemessen erscheinen, Objekte als ähnlich zu betrachten, die übereinstimmend viele Eigenschaften nicht aufweisen, in anderen wäre dies geradezu absurd. In der Domäne der Fortbewegungsmittel könnte das Fehlen eines Motors durchaus Aufschluß auf die Ähnlichkeit zweier Objekte geben, während der Schluß, der Löwe und die Gans seien sich ähnlich, weil sie beide kein gestreiftes Fell haben, doch recht gewagt ist. [Sok73] kommen nach ausgiebiger Analyse zu dem Schluß, daß es keine Möglichkeit gibt, schnell und valide ohne ausgeprägte Kenntnis der Daten eine Entscheidung darüber zu treffen, ob der Wert d bei der Analyse der jeweils zu gruppierenden Daten zu berücksichtigen sei oder nicht.

In [Eve80] finden sich die in Abbildung 3.1 aufgeführten Auswertungen der Parameter, wobei das *einfache Matching* (i) das gebräuchlichste Maß darstellt, welches d berücksichtigt; *Jaccard's Koeffizient* (ii) wird am häufigsten verwendet, wenn d nicht in die Ähnlichkeitsbestimmung eingehen soll.

Offensichtlich führen die unterschiedlichen Formeln zu unterschiedlichen Ähnlichkeiten. Wichtig ist festzuhalten, daß sich dies nicht monoton verhält: Seien etwa s_i, s_j zwei Formeln aus Abbildung 3.1 mit $s_i(X_1, Y_1) > s_i(X_2, Y_2)$ und $s_j(X_1, Y_1) > s_i(X_1, Y_1)$, so kann daraus nicht gefolgert werden, daß auch $s_j(X_1, Y_1) > s_j(X_2, Y_2)$ gilt. Das wird in der Regel für Objekte X_1, X_2, Y_1, Y_2 zur

Konsequenz haben, daß unterschiedliche Formeln zu unterschiedlichen Gruppierungen führen, da z. B. bei Verwendung der einen die Objekte X_1, Y_1 am ähnlichsten sind, während eine andere etwa das Paar X_2, Y_2 vorzieht.

Sind die Attributwerte der betrachteten Objekte nicht binär, sondern symbolischer Natur (**nominale** Attributwerte), so bietet es sich für eine recht geringe Gesamtanzahl unterschiedlicher Attributwerte an, die qualitativen Werte in eine binäre Form zu überführen, um obige Auswertung zu ermöglichen. Dabei führt jeder mögliche Attributwert zu einem binären Attribut: Sei etwa das nominale Attribut $Haarfarbe$ mit dem Wertebereich $\{blond, schwarz, rot\}$ gegeben, so werden daraus 3 Attribute $(Haarfarbe_{blond}, Haarfarbe_{schwarz}, Haarfarbe_{rot})$, mit jeweils einem Wert aus $\{0, 1\}$. Es ist offensichtlich, daß dieses Vorgehen zu einer erhöhten Anzahl von Attributen führt, die zudem korreliert sind; auf der anderen Seite sorgt es dafür, daß wir uns im sicheren Gebiet der gut erforschten Koeffizienten für binäre Attribute bewegen können. Es sollte dabei dann in jedem Fall eines der in Abbildung 3.1 angegebenen Maße benutzt werden, das den Wert d (siehe Abbildung 3.2) berücksichtigt, da auf diesem Weg sehr natürlich eine Gewichtung vorgenommen wird, die der Intuition entspricht: Weisen zwei Objekte eine 1 in einem durch die Übersetzung gewonnenen Attribut (z. B. $Haarfarbe_{rot}$) auf, so weisen sie übereinstimmend in den zugehörigen anderen Attributen $(Haarfarbe_{blond}, Haarfarbe_{schwarz})$ eine 0 auf. Das führt dazu, daß eine Gleichheit in einem nominalen Attribut mit großem Wertebereich höher bewertet wird, als eine Gleichheit in einem mit einer kleinen Wertemenge.

Verbietet eine zu große Attributwerteanzahl diese Überführung, so bleibt zunächst nur die Möglichkeit, die Ähnlichkeit trivial über die (Un-)Gleichheit der einzelnen Komponenten zu bestimmen[4]:

$$s(X_k, Y_k) := 1 \text{ falls } X_k = Y_k,\ 0 \text{ sonst}$$

Für quantitative Daten (**numerische** Attributwerte) wird in der Regel ein Vertreter der Minkowski-Metren

$$d(X, Y) = (\sum_{k=1}^{p} |X_k - Y_k|^L)^{\frac{1}{L}}$$

mit $L \geq 1$ eingesetzt. $L = 1$ führt zu der Hamming- oder auch *City-Block-Metrik*, $L = \infty$ zum Chebyshev-Abstand[5]. Am häufigsten wird mit $L = 2$ der *euklidische Abstand* verwendet, wobei $d(X, Y)$ gemäß obiger Ausführung in einen Ähnlichkeitswert überführt werden kann. Während die zuvor beschriebenen Maße voraussetzten, daß die untersuchten Daten insofern homogen sind,

[4] Im Abschnitt 3.3.3 wird die Möglichkeit beschrieben, Teilergebnisse von CLUST_DB hierfür heranzuziehen.

[5] Siehe etwa [Mur92].

daß die Attribute entweder alle binär oder alle nominal sind, können hier symbolische und binäre Attributwerte etwa durch die naheliegende Festlegung

$$X_k - Y_k := 0 \text{ falls } X_k = Y_k,\ 1 \text{ sonst}$$

ebenfalls verarbeitet werden.

Neben dieser vorteilhaften Eigenschaft weist dieses Maß eine recht unangenehme Eigenschaft auf: Es ist stark von der Skalierung der numerischen Attribute abhängig. Ein offensichtlicher Nachteil, der zusätzliche Arbeit erfordert und dadurch unangenehm wird, daß in der Literatur Uneinigkeit darüber herrscht, durch welche Art von Standardisierung dieser „Schaden" behoben werden sollte. Am häufigsten wird der Ansatz der *Normalisierung* durch Subtraktion einer Konstanten, so daß der Mittelwert Null beträgt und *Reduktion* mittels Division durch die Standardabweichung verfolgt, während [Coo88] in einer vergleichenden Studie die Division durch den Wertebereich favorisieren.

Das letzte im Rahmen der Objektähnlichkeiten vorzustellende Maß ist ein Ähnlichkeitsmaß, das von [Gow71] gezielt für den Einsatz bei **gemischten Datentypen** eingeführt wurde:

Der Ähnlichkeitswert $s(X, Y)$ für zwei Objekte X, Y berechnet sich als gewichtete[6] Summe von Ähnlichkeitskoeffizienten $s_k(X, Y)$, die für jedes Attribut k in Abhängigkeit des Datentyps separat bestimmt werden:

(A) binär: $s_k(X, Y) = 1$ für $X_k = Y_k = 1$, 0 sonst

(B) nominal: $s_k(X, Y) = 1$ falls $X_k = Y_k$, 0 sonst

(C) numerisch: $s_k(X, Y) = 1 - |X_k - Y_k|/R_k$

Man erkennt, daß dieses Maß für (A) dem Koeffizienten (ii) aus Abbildung 3.1 entspricht und im Bereich (C) im Gegensatz zum euklidischen Abstand für eine Form von Standardisierung durch Betrachtung des Wertebereiches R_k gesorgt wird.

Keines der genannten Objektkonzepte kann ohne weiteres für die Lösung der Aufgabenstellung herangezogen werden, da in ihnen verankert ist, daß das Objekt durch ein *Tupel* beschrieben wird. Wir benötigen hingegen ein Maß, das einen *Attributwert* als Objekt auffassen kann. Auf der anderen Seite ist es genau diese veränderte Ausgangsposition, die im vorliegenden Fall einen Cluster-Ansatz erst ermöglicht: Schon allein das Aufstellen einer Ähnlichkeitsmatrix für eine reale Datenbank mit 250.000 Tupeln ist undurchführbar, während die Anzahl der unterschiedlichen Attributwerte eines nominalen Attributs üblicherweise um Größenordnungen kleiner ist als die Gesamtzahl der Tupel. In Abschnitt 3.2.2 werden ein taugliches Objekt–Objekt-Maß definiert und die Beziehung zu einer Objektrepräsentation geklärt, die den Einsatz der beschriebenen Maße ermöglicht.

[6] Das Gesamtgewicht ist $1/w$, wobei sich w durch Summation der $w_k(X, Y)$ ergibt. Diese sind 1, wenn für beide Objekte ein Wert für Attribut k vorliegt, sonst 0.

Gruppenähnlichkeit

Es werden im folgenden verschiedene Ansätze wiedergegeben, die zuvor beschriebenen Festlegungen von Ähnlichkeiten bzw. Distanzen dahingehend zu erweitern, daß auch zwei Gruppen miteinander verglichen werden können.

Eine bei numerischen Daten naheliegende Vorgehensweise ist es, den Abstand zweier Gruppen als euklidischen Abstand der beiden Gruppenmittelwertvektoren $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_p)$, $\overline{y} = (\overline{y}_1, \ldots, \overline{y}_p)$ zu definieren:

$$d(X,Y) := \sqrt{\sum_{k=1}^{p} (\overline{x}_k - \overline{y}_k)^2}$$

Eine andere Herangehensweise stellt der Ansatz dar, die Gruppenähnlichkeit $s(X,Y)$ unmittelbar aus den einzelnen Objektähnlichkeiten zu gewinnen:

Beim *nearest neighbour* zieht man dazu das Objektpaar aus $X \times Y$ heran, das den kleinsten Abstand bzw. die größte Ähnlichkeit aufweist und definiert dies als Distanz / Ähnlichkeit der beiden Gruppen. Diese Festlegung wird beispielsweise im Cluster-Verfahren Single Linkage [KF51] genutzt. Da es für einen Zusammenschluß zweier Cluster genügt, wenn sie (bei geometrischer Interpretation) an einer Stelle benachbart sind, kann es zum möglicherweise ungewünschten *chaining* kommen, das langgezogene, wenig kompakte Cluster induziert.

Analog zu obigem kann auch das entfernteste bzw. unähnlichste Objektpaar den gesuchten Wert festlegen. Man spricht dann vom *furthest neighbour*, der beim Cluster-Verfahren Complete Linkage [Eve80] Verwendung findet.

Neben diesen beiden zuletzt genannten — im gewissen Sinne extremen — Ausprägungen bietet sich als Kompromiß geradezu an, nicht nur ausschließlich den nächsten bzw. entferntesten Nachbarn zu betrachten, sondern einen Durchschnitt über alle Objektpaare aus $X \times Y$ zu bilden, um diesen dann als Gruppenwert zu definieren. Dies bildet die Grundlage für das Verfahren Group Average [Wil66].

Im Falle von Ähnlichkeiten, läßt sich obiges formal wie folgt zusammenfassen[7]:

nearest neighbour: $s(X,Y) := max\{s(i,j) | i \in X, j \in Y\}$

furthest neighbour: $s(X,Y) := min\{s(i,j) | i \in X, j \in Y\}$

average: $s(X,Y) := \frac{1}{|X| \cdot |Y|} \sum_{i \in X, j \in Y} s(i,j)$

Neben der Tatsache, daß die Aktualisierung der Matrix bei der Benutzung eines dieser drei Verfahren wenig aufwendig ist, bieten Single und Complete Linkage den weiteren Vorteil, daß sie invariant gegenüber Reihenfolgeerhaltenden Transformationen sind: Da sie jeweils nur den größten bzw. den kleinsten Ähnlichkeitswert auswählen, sind sie recht robust gegenüber Datenfehlern, die zu einer Verfälschung der Matrix führen.

[7] Für Distanzen müssen dual *max* und *min* beim nearest und furthest neighbour ersetzt werden.

Einen sehr interessanten weil flexiblen Ansatz bietet die Lance-Williams-Rekursionsformel [Wil67]:

$$d(X_1 \cup X_2, Y) := \alpha_1 \cdot d(X_1, Y) + \alpha_2 \cdot d(X_2, Y) + \beta \cdot d(X_1, Y_1) + \gamma \cdot |d(X_1, Y) - d(X_2, Y)|$$

Durch entsprechende Wahl der Werte für $\alpha_1, \alpha_2, \beta, \gamma$ erhält man unter anderem einige der bereits vorgestellten Maße:

nearest neighbour: $\alpha_1 = \alpha_2 = 1/2$; $\beta = 0$; $\gamma = -1/2$

furthest neighbour: $\alpha_1 = \alpha_2 = 1/2$; $\beta = 0$; $\gamma = 1/2$

average: $\alpha_1 = |X_1|/(|X_1| + |X_2|)$; $\alpha_2 = |X_2|/(|X_1| + |X_2|)$; $\beta = \gamma = 0$

Während die Gruppe–Gruppe-Maße nearest neighbour, furthest neighbour und average universell einsetzbar und damit für die vorliegende Problematik interessant sind, scheidet der euklidische Abstand der Gruppenmittelwerte aus, da dies bei symbolischen Werten keinen Sinn ergibt.

Es sollte in diesem Abschnitt deutlich geworden sein, daß es der Überlegung bedarf, welches Maß das „rechte“ (für die jeweilige Anwendung) darstellt. Es ist unbestritten, daß kein in allen Domänen perfektes Maß bekannt ist. B. Everitt weist in Bezug auf die Wahl des Ähnlichkeitsmaßes auf eine grundsätzliche Zirkularität hin:

> „Die Wahl des Maßes würde sehr durch die Kenntnis der Struktur der Daten erleichtert — aber genau diese Struktur soll mit der Cluster-Analyse aufgedeckt werden. Wenn wir die Gruppierungen hätten, könnten wir das angebrachte Maß benennen, wüßten wir um das passende Maß, ergäben sich die gewünschten Gruppierungen“ [Eve80, S.22].

3.1.3 Cluster-Techniken

Nachdem der Schritt der Überführung der Objektbeschreibungen ($n \times p$ - Matrix) in eine *proximity*-$n \times n$-Matrix durch Wahl eines Maßes und entsprechender Rechnungen vollzogen wurde, gilt es festzulegen, welche Technik verwendet werden soll, um letztendlich die Gruppierungen zu extrahieren.

[Eve80] unterscheidet im wesentlichen zwischen hierarchischen Verfahren, Optimierungsverfahren, *density*- oder auch *mode seeking*- Verfahren und *clumping*. Letzteres wird nicht näher betrachtet werden, da dieses Konzept Überlappungen der Cluster zuläßt und sich somit nicht für die Typinduktion anbietet, die eine Partitionierung der Werte erfordert.

Hierarchische Verfahren

Wie der Name bereits verheißt, erzeugen hierarchische Verfahren eine Objekthierarchie, die typischerweise als binärer Baum bzw. *Dendogramm* visualisiert wird. Formal läßt sich das Ergebnis eines hierarchischen Verfahrens wie folgt beschreiben:

Definition 3.3 *Sei I die Menge der Objekte und $\mathcal{P}(I)$ die entsprechende Potenzmenge. $H \subseteq \mathcal{P}(I)$ ist eine* **Objekthierarchie** *genau dann, wenn die folgenden Punkte erfüllt sind:*

(1) $I \in H$

(2) $\forall x \in I : \{x\} \in H$

(3) $\forall h, h' \in H : (h \cap h' = \emptyset) \vee (h \subset h') \vee (h' \subset h)$

Hierarchische Verfahren unterscheiden sich grundsätzlich darin, ob sie die Hierarchie von der Spitze herunter *top-down* oder von unten nach oben *bottom-up* aufbauen. Verfahren, die letztere Richtung einschlagen, werden im Englischen *agglomerative* genannt, die anderen *divisive*.

Die bottom-up Verfahren erfreuen sich großer Beliebtheit, da ihr Konzept in der Regel leicht umzusetzen ist: Es werden solange alle Objekte betrachtet und die ähnlichsten beiden zu einem neuen Objekt zusammengeführt, bis nur noch ein Cluster übrig ist, der alle ursprünglichen Objekte enthält. Bezüglich der Frage, wie die Ähnlichkeiten im Laufe des Prozesses bestimmt bzw. bei der Fusion zweier Objekte aktualisiert werden, sei auf den vorangegangenen Abschnitt verwiesen.

Top-down Verfahren erfordern im Gegensatz dazu eine ausgeklügelte Heuristik (*control strategy*), die hoffen läßt, aus den exponentiell vielen Möglichkeiten, eine Menge in zwei Teilmengen zu partitionieren, bzgl. einer *objective function* eine möglichst gute Wahl zu treffen, ohne alle Möglichkeiten durchzurechnen. Daß einmal getroffene Entscheidungen bzgl. der Auftrennung oder der Fusion bei hierarchischen Verfahren unwiderruflich sind, stellt deshalb insbesondere für diese Verfahren ein echtes Problem dar. Als vorteilhaft kann es sich jedoch bei geschickter Wahl eines entsprechenden Kriteriums erweisen, daß ein top-down Verfahren an geeigneter Stelle die Unterteilung der Gruppen stoppen kann.

Bei den top-down Verfahren wird zwischen zwei verschiedenen Konzepten unterschieden: *Polythetisch* bedeutet, daß alle Attribute die anstehende Aufspaltung beeinflussen, während *monothetische* Verfahren in jedem Schritt jeweils ein Attribut auswählen, das allein den jeweiligen Aufteilungsprozeß bestimmt. Zur Wahl dieses Attributs werden in den meisten Fällen probabilistische bzw. informationstheoretische Berechnungen herangezogen. Diese Herangehensweise läßt sich in einem anderen wichtigen Bereich des maschinellen Lernens wiederfinden: ID3 [Qui86] als bekanntestes Mitglied der Familie der *TDIDT*[8]-Algorithmen geht bei der Lösung der Aufgabe, einen Entscheidungsbaum anhand klassifizierter Beispiele zu erzeugen, analog vor. In jedem Schritt wird dabei ein Attribut ausgewählt, das bei einer Aufspaltung der Beispiele anhand seiner Werte den größten Informationsgewinn in Bezug auf die Separation der unterschiedlichen Klassen liefert.

[8]TDIDT: Top-Down Induction of Decision Trees

Gemein ist den hierarchischen Verfahren, daß bei ihrer Verwendung aus der Menge der möglichen Partitionen — jede Ebene der Hierarchie stellt eine dar — die für die Anwendung nützlichste auszuwählen ist. Im einfachsten Fall wird diese Entscheidung dem Benutzer überlassen, der auf diesem Wege auch sein Wissen über den Sachbereich mit einbringen kann.

Optimierungsverfahren

Optimierungsverfahren sind darauf ausgerichtet, vom Benutzer vorgegebene oder selbst erzeugte initiale Cluster bzgl. eines numerischen Kriteriums zu verbessern. Sie liefern (in der Regel nichthierarchische) Partitionen, weshalb sie auch häufig unter dem Stichwort *partitioning* geführt werden. Bei ihnen kann im Gegensatz zu den hierarchischen Verfahren eine ursprünglich erfolgte Zuordnung eines Objektes zu einem Cluster rückgängig gemacht werden, wenn sie sich im Laufe der Optimierung als unvorteilhaft erweist.

Verfahren, die diesem Ansatz folgen, lassen sich zum einen anhand des verwendeten numerischen Optimierungskriteriums und zum anderen anhand der Methode, die benutzt wird, um initiale Cluster zu erzeugen, charakterisieren. Während die Initialisierung der k Cluster zumeist nicht besonders erwähnenswert ist — es werden oft die k ersten, die k von einander entferntesten oder k zufällige Beobachtungen ausgewählt — geben die Optimierungskriterien mehr her: Bei den meisten von ihnen steht die in der Statistik bekannte Matrix-Gleichung $T = W + B$ im Mittelpunkt, wobei sich die einzelnen Matrizen, deren Bezeichnungen sich aus dem Englischen ableiten, die folgende Bedeutung haben[9]: T ist Matrix der Gesamtvarianz (*total dispersion matrix*), W die intra (*within*) Gruppenvarianz — summiert über alle k Gruppen — und B die inter (*between*) Gruppenvarianz. Da T konstant ist, sind die beiden möglichen Optimierungsstrategien (Minimierung von W, Maximierung von B) äquivalent zueinander.

Während der einfache Spezialfall $p = 1$ zu einer skalaren Gleichung degeneriert, die eine Optimierungsprozedur offenbart (siehe [Fis58]), ist im allgemeinen Fall $p > 1$ eine solche nicht offensichtlich. Deshalb konzentriert man sich häufig auf die Minimierung der Determinante von W und noch häufiger auf die Minimierung der *Spur* von W: Aus obiger Gleichung folgt $Spur(T) = Spur(W) + Spur(B)$. Diesem Ansatz folgen in letzter Konsequenz auch Verfahren, die die Objekte so lange unterschiedlich zuordnen, bis alle dem Zentrum ihrer Gruppe (euklidisch) näher sind, als einem Zentrum irgendeiner anderen Gruppe (siehe nachfolgendes Beispiel).

Der bekannteste Repräsentant dieser Klasse von Verfahren ist das *k-Means Clustering* [Mac67], welches kurz in einer Variante beschrieben wird: Der Algorithmus sucht nach einer Partitionierung der Beobachtungen in eine vom Benutzer anzugebenden Anzahl k von Clustern, die jeweils durch ein Cluster-Zentrum (*mean*) beschrieben werden. Gestartet wird der Vorgang mit k zufällig gewählten Beobachtungen, die als initiale Zentren dienen. Im weiteren Verlauf werden alle Beobachtungen der Reihe nach dem jeweils nächsten Cluster-Zentrum

[9] Die ausführlichen Definitionen der Matrizen lassen sich etwa in [Wil62] finden.

zugeordnet, wobei die Lage des Zentrums sich in die Richtung der neuen Beobachtung „bewegt“. Je mehr Beobachtungen sich in dem Cluster befinden, desto „schwerer“ ist er und desto kleiner fällt die Bewegung aus. Das Verfahren stoppt, nachdem alle Beobachtungen abgearbeitet wurden.[10]

Neben den Verfahren, die auf der genannten Matrix-Gleichung beruhen, gibt es auch solche, die ein Informationsmaß benutzen (z. B. SNOB [Dow94]) oder eine Cluster-Stabilität definieren, deren Durchschnitt über alle Cluster zu maximieren ist.

Auch wenn einige Verfahren dynamisch die Anzahl k der Cluster verändern können, so stellt die Wahl dieses Parameters doch in den meisten Fällen ein Problem dar — vergleichbar mit der Schwierigkeit, die für eine Partition günstigste Ebene eines hierarchischen Verfahrens zu bestimmen. Dies wird in der Regel dazu führen, daß der Benutzer eine Reihe von unterschiedlichen Werten für k ausprobieren wird — in der Hoffnung, daß die Ergebnisse einen Schluß über die angemessene Wahl nahelegen. Des weiteren können Optimierungsverfahren keine global-optimale Lösung garantieren. Sie erreichen lediglich ein lokales Maximum, was ebenfalls mehrere Lernläufe mit unterschiedlichen Startkonfigurationen nötig macht.

Density Search

Eine geometrische Interpretation der Cluster-Analyse läßt die einzelnen Objekte zu Punkten des p-dimensionalen durch die unterschiedlichen Attributwerte aufgespannten Raums werden. Dies wird besonders anschaulich, beschränkt man die Anzahl der Attribute p auf 3.

Es ist bei dieser Betrachtung ein natürliches Konzept, nach Gebieten zu suchen, die *dicht* von Punkten besetzt sind und von einer *wenig dichten* Zone umschlossen werden, um diese Gebiete als Cluster zu extrahieren. Viele dieser „nach Dichte suchenden“ Verfahren erwuchsen dem Versuch, SINGLE LINKAGE dahingehend zu verbessern, das *chaining* — die Tendenz, Punkte eher in ein existierenden Cluster aufzunehmen als eine neue Gruppe zu erzeugen — zu entschärfen.[11] So werden bei der *Taxmap-Methode* [Sne69] dem SINGLE LINKAGE zusätzliche Kriterien zur Seite gestellt, die entscheiden, wann das Hinzufügen zu einem Cluster gestoppt werden sollte: Ein nearest neighbour wird etwa nur dann aufgenommen, wenn die Differenz von resultierender neuer durchschnittlicher Ähnlichkeit der Gruppenmitglieder und Absinken der durchschnittlichen Ähnlichkeit weniger als C beträgt.

[Lev70] bauen bei ihrer Methode um unimodale Fuzzy-Mengen[12] zu finden, ebenfalls auf SINGLE LINKAGE auf, betrachten jedoch die zu beurteilenden Punkte in einer Reihenfolge, die auf dem *grade of membership* der Punkte beruht. Dieser gibt an, wieviele Punkte sich im Umkreis T von x_i in Bezug auf

[10] Eine andere, weniger von der Reihenfolge abhängige aber aufwendigere Variante, aktualisiert die Cluster-Zentren erst dann, wenn alle Beobachtungen verteilt wurden. Danach wird solange der gesamte Prozeß wiederholt, bis sich keine Veränderungen mehr bzgl. der Cluster ergeben.

[11] Zu SINGLE LINKAGE und *chaining* siehe Abschnitt 3.1.2.

[12] Zur Definition von Fuzzy-Mengen siehe [Zad65].

den euklidischen Abstand befinden:

$$gom(X_i) := |\Gamma_i| = |\{x | d(x_i, x) \leq T\}|$$

So anschaulich das verfolgte geometrische Konzept auch ist, so machen diese beiden Vertreter schon deutlich, daß die untrennbar mit der Cluster-Analyse verbundene Frage nach dem Maß letztendlich nur auf die Festlegung von Parametern verlagert wird. Bei der Taxmap-Methode ist dies der Schwellwert C als Indikator für (Dis-)Kontinuität, während es bei der Entdeckung der unimodalen Fuzzy-Mengen eines vom Benutzer anzugebenden Radius T bedarf.

Wie bereits zu Beginn dieses Abschnittes bemerkt wurde, spielt es für die Cluster-Techniken ganz im Gegensatz zu den Ähnlichkeitsmaßen auf Objekt—Objekt-Ebene eine untergeordnete Rolle, welche Struktur die Objekte tragen. Es läßt sich damit abschließend kurz und knapp festhalten, daß die vorgestellten Techniken für die Bearbeitung der Aufgabe in Frage kommen.

3.2 Clust_DB

In diesem Abschnitt wird der Ansatz vorgestellt, der dem im Rahmen dieser Diplomarbeit entwickelten Programm CLUST_DB zugrunde liegt. Es handelt sich dabei um einen hierarchischen bottom-up Cluster-Ansatz, dessen Komponenten der Reihe nach vorgestellt werden:

Nachdem zu Beginn des dritten Kapitels bereits der Einsatz der Cluster-Analyse erläutert wurde, wird zunächst im Abschnitt 3.2.1 die Wahl der agglomerativen hierarchischen Cluster-Technik begründet werden. Der darauf folgende Abschnitt wird sich mit dem verwendeten Ähnlichkeitsmaß befassen, während Abschnitt 3.2.3 der Nullwertbehandlung von CLUST_DB zugedacht ist.

Abschnitt 3.2.4 wird einen Überblick über das Verfahren bieten, der eine Komplexitätsabschätzung beinhaltet. Im Abschnitt 3.2.5 wird die algorithmische Ebene verlassen, um Details der Implementation zu erörtern. Gegenstand des abschließenden Abschnitts 3.2.6 wird die Wahl und Gewichtung der für das Maß heranzuziehenden Attribute sein.

3.2.1 Cluster-Technik

CLUST_DB liefert als Ergebnis eine Cluster-Hierarchie, die bottom-up aufgebaut wird. Die hierarchische Methode besitzt den Vorteil, daß für den Aufbau der Gruppen im Gegensatz zu den meisten anderen Verfahren keine vom Benutzer vorzugebenden Parameter nötig sind. Besonders im Hinblick auf die Datenmenge realer Datenbanken erschien es wichtig, Durchgänge, die für die Bestimmung angemessener Parameter erforderlich gewesen wären, einzusparen.

Der Nachteil, daß die eigentlich gesuchten Partitionen der Hierarchie entnommen werden müssen, wog nicht so schwer: Die Strukturinformation einer Hierarchie dürfte in den meisten Fällen für diesen erhöhten Aufwand entschädigen, so daß meiner Meinung nach dies hier nicht als gravierender Nachteil hierarchischer Verfahren bezeichnet werden sollte. So können Lernverfahren wie etwa INDUCE [Mic83] und OTIS [Ker88] diese Information direkt nutzen. Insbesondere für die Rolle CLUST_DBs als eigenständiges *KDD-tool* (siehe Abschnitt 3.3) leistet sie einen Beitrag.

Daß die Wahl auf die bottom-up Herangehensweise fiel, begründet sich darin, daß die aufwendigeren top-down Verfahren — man denke dabei an die *control strategies* — hier ihren Vorteil gegenüber den bottom-up Verfahren nicht richtig entfalten können: Ein Stopp an der falschen Stelle hätte einen neuen Lernlauf erfordert und damit den eigentlichen Vorteil mehr als zunichte gemacht.

3.2.2 Ähnlichkeitsmaß

Im folgenden wird das Kernstück von CLUST_DB, das verwendete Ähnlichkeitsmaß, vorgestellt. Gemäß der Ausführungen in Abschnitt 3.1.2 werden zunächst die Objekt- und dann die Gruppenähnlichkeit definiert.

Objektähnlichkeit

In der Diskussion der in der Cluster-Analyse gebräuchlichen Ähnlichkeitsmaße auf der Objektebene ist dargestellt worden, daß sie auf der Beschreibung eines Objektes durch ein Tupel von Attributwerten basieren. Eine entsprechende Ausgangssituation ist hier nicht ohne weiteres gegeben: Alles, was wir zunächst haben, sind die Attributwerte des zu gruppierenden Attributs.

Eine triviale Möglichkeit besteht darin, aus den Häufigkeiten, mit denen zwei Attributwerte in der Tabelle zu finden sind, die Ähnlichkeit dieser beiden abzuleiten. Dieser Ansatz wurde als zu schwach bewertet und nicht weiter verfolgt.

Vielversprechender ist der Versuch, eine Objektrepräsentation zu finden, die den Einsatz der vorgestellten Maße ermöglicht. Ein erster Versuch könnte wie folgt aussehen: Für den Fall, daß jedes Objekt (d. h. jeder Attributwert) a_i des zu gruppierenden Attributs A nur einmal in der Datenbank vorkommt, könnte er etwa durch das Tupel $(b_{i_1}, \ldots, b_{i_n})$ mit $(a_i, b_{i_1}, \ldots, b_{i_n}) \in R$ dargestellt werden und die Berechnung der Ähnlichkeiten gemäß eines der vorgestellten Maße realisiert werden.

Eine plausible Erweiterung für den allgemeineren (und sehr viel wahrscheinlicheren) Fall, daß a mehrfach als Attributwert von A vorkommt, ist es dann, alle entsprechenden Tupel zu einer Menge zusammenzufassen und im folgenden die bekannten Maße so zu verwenden, als handele es sich dabei um bereits gebildete Cluster. Diese Repräsentation — im weiteren Verlauf als *Tupelrepräsentation* bezeichnet — hat allerdings den Haken, daß mit allen Tupeln der Datenbanktabelle gearbeitet werden muß:

Lemma 3.1 *A sei das zu gruppierende Attribut und $B_1 \ldots B_n$ die übrigen in der Tabelle R vorkommenden Attribute. Der Platzbedarf für die Tupelrepräsentation der Objekte kann auf $\mathcal{O}(m_A \cdot {m_{B_{max}}}^n)$ anwachsen, wobei $m_{B_{max}}$ die maximale Anzahl verschiedener vorkommender Attributwerte von $B_1 \ldots B_n$ und m_A die Anzahl verschiedener vorkommender Werte für A bezeichnet.*[13]

[13] Es ist also $m_A := |\pi_A(R)|$ und $m_{B_{max}} := max\{|\pi_{B_1}(R)|, \ldots, |\pi_{B_n}(R)|\}$.

Beweis Die Mengen, die als Repräsentation für ein Objekt dienen, enthalten n-Tupel. Da es für jede der n Stellen des Tupels unabhängig voneinander bis zu $m_{B_{max}}$ Möglichkeiten gibt, sie zu besetzen, folgt, daß jede der m_A Mengen bis zu ${m_{B_{max}}}^n$ Elemente enthalten kann. □

Niemand wird versuchen wollen, eine reale Datenbank in den Speicher zu holen, so daß die Anzahl der Datenbankzugriffe für die Cluster-Analyse sehr groß werden dürfte.

Diese Vorüberlegungen führten zu dem verwendeten Maß, das im folgenden vorgestellt wird. Es wird zunächst formal eingeführt, um es anschließend ausführlich zu diskutieren.

Definition 3.4 *Das von* CLUST_DB *verwendete Maß für die* **Ähnlichkeit zweier Attributwerte** a_i, a_j *des zu gruppierenden Attributs* A *einer Tabelle* R *einer relationalen Datenbank ist definiert als*

$$(1) \quad s(a_i, a_j, A, R) := \sum_{k=1}^{n} w_k \cdot s_{B_k}(a_i, a_j, A, R)$$

wobei $B_1 \dots B_n$ *vom Benutzer gewählte Attribute von* R*,* $w_1 \dots w_n$ *deren zugeteilte Gewichte sind und* $s_{B_k}(a_i, a_j)$ *als Ähnlichkeit von* a_i *und* a_j *bzgl. des Attributs* B_k *wie folgt definiert ist:*

$$(2) \quad s_{B_k}(a_i, a_j, A, R) := \frac{|M_{B_k}(a_i, A, R) \cap M_{B_k}(a_j, A, R)|}{|M_{B_k}(a_i, A, R) \cup M_{B_k}(a_j, A, R)|}$$

mit

$$(3) \quad M_{B_k}(a_i, A, R) := \pi_{B_k}(\sigma_{A=a_i}(R)),$$

deren Elemente im folgenden auch als Mitspieler von a_i *bzgl.* B_k *bezeichnet werden.*

Diese Definition soll nun näher erläutert werden. Da aus dem Kontext stets hervorgeht, welches Zielattribut A welcher Tabelle R betrachtet wird, wird im folgenden auf deren Angabe der Übersichtlichkeit halber verzichtet.

(1) verlagert das Problem zunächst auf (2), indem festgestellt wird, daß sich die *Gesamtähnlichkeit* $s(a_i, a_j)$ als gewichtete Summe des eigentlichen Maßes $s_{B_k}(a_i, a_j)$ berechnet. (3) dient dazu, (2) etwas übersichtlicher werden zu lassen. Zudem wird die kombinierte Operation der Projektion einer Selektion[14] $\pi_{B_k}(\sigma_{A=a_i}(R))$, die die Menge $\{v | (a_i, v) \in \pi_{AB_k}(R)\}$ beschreibt, durch $M_{B_k}(a_i)$ abgekürzt.

(3) verdeutlicht aber auch, worauf sich das Maß stützt: Das Objekt $a_i \in \pi_A(R)$ wird durch ein n-Tupel repräsentiert, dessen Komponenten die jeweiligen *Mitspieler* von a_i bilden. Es werden zwei Werte (bzgl. eines Attributs B_k) als ähnlich betrachtet, wenn sie (bzgl. B_k) etwa die gleichen Wertemengen aufweisen. Diesem Ansatz liegt die Annahme zugrunde, daß die Attributwerte der unterschiedlichen Attribute nicht völlig unabhängig voneinander sind, sondern

[14] Siehe hierzu Kapitel 2.

vielmehr semantisch zusammengehören. Im folgenden werden A auch als *Ziel-* und $B_1 \dots B_n$ als *Quellattribute* bezeichnet.

Die Ähnlichkeit mit dem zu Beginn gemachten naiven Tupel-Ansatz ist offensichtlich. Der Unterschied scheint subtil, hat aber weitreichende Konsequenzen, wie die dem Beispiel folgende Analyse der *Mitspielerrepräsentation* in Bezug auf den Platzbedarf zeigen wird.

Zunächst aber soll ein kleines Beispiel basierend auf Tabelle 2.1 den Vorgang $Objekt \rightarrow Objektrepräsentation \rightarrow Ähnlichkeitswert$ verdeutlichen. Dabei sei $NAME$ in Tabelle $LIEFERANT$ als zu gruppierendes Attribut gewählt und die Darstellung auf das Quellattribut $WARE$ mit $w_{WARE} := 1$ beschränkt.

Beispiel 3.1

$$\begin{aligned}
M_{Ware}(Bauer - Bill) &= \{Kartoffeln\} \\
M_{Ware}(Farmer - Joe) &= \{Geflügel, Gemüse\} \\
M_{Ware}(Fischers - Fritze) &= \{Fisch\} \\
M_{Ware}(Hof - Holt) &= \{Gemüse, Obst\} \\
M_{Ware}(Mc - Dorado) &= \{Fisch, Mais, Geflügel\}
\end{aligned}$$

$$\begin{aligned}
s_{Ware}(Bauer - Bill, Farmer - Joe) &= 0/3 \quad \dots \\
s_{Ware}(Farmer - Joe, Hof - Holt) &= 1/3 \\
s_{Ware}(Farmer - Joe, Mc - Dorado) &= 1/4 \quad \dots \\
s_{Ware}(Hof - Holt, Mc - Dorado) &= 0/5
\end{aligned}$$

Insbesondere die Mitspielermengen dürften durch das Beispiel an Anschaulichkeit gewonnen haben, so daß das folgende Lemma keine Schwierigkeit bereiten sollte:

Lemma 3.2 *Der Platzbedarf für die Mitspielerrepräsentation der Objekte ist durch $\mathcal{O}(n \cdot m_A \cdot m_{B_{max}})$ beschränkt, wobei $m_{B_{max}}$ die maximale Anzahl verschiedener vorkommender Attributwerte von $B_1 \dots B_n$ und m_A die Anzahl verschiedener vorkommender Werte für A bezeichnet.*

Beweis Ein Objekt a_i wird durch ein n-Tupel beschrieben, dessen Stellen durch $M_{B_k}(a_i)$, die Menge der Mitspieler bezüglich des Attributs B_k, besetzt werden. Jede dieser n Mengen kann maximal $m_{B_{max}}$ Elemente besitzen, was für jedes der m_A Objekte zu einem Platzbedarf von $\mathcal{O}(n \cdot m_{B_{max}})$ führt. □

Bei der aufgezeigten Datenreduktion muß man sich selbstverständlich fragen, zu welchem Preis diese erkauft wurde, welche Konsequenzen dies hat. Eine Reduktion ist letztendlich nichts wert, wenn ein großer Teil der Information, die in den Daten steckt, verloren geht. In der Repräsentation über die Mitspielermengen steckt nicht die Information, in welchen Kombinationen die Attributwerte vorkamen. Auch kann sie wegen der Mengeneigenschaft keine Auskunft mehr über die Anzahl der unterschiedlichen Attributwerte geben. Während beide „Verluste" bei der Ähnlichkeitsberechnung keine Rolle spielen, macht letzterer CLUST_DB anfälliger in Bezug auf Datenfehler: Ein falscher Attributwert b_{noise} in einer Mitspielerliste wiegt genauso schwer, wie ein hundertfach gelesener (korrekter) Wert b_{good}. Multimengen stellen keine Alternative dar, da sie den gleichen Platzbedarf wie die Tupelrepräsentation benötigen. Im Abschnitt 3.6.3 wird dieser Punkt aufgegriffen.

Gruppenähnlichkeit

Im Fazit des Abschnitts 3.1.2 über die bekanntesten Ähnlichkeitsmaße für den Vergleich zweier Gruppen von Objekten wurde bereits festgehalten, daß einige von ihnen — ganz im Gegensatz zu den Maßen auf der Objektebene — nicht von der Objektrepräsentation abhängen. So wurden etwa *furthest* bzw. *nearest neighbour* und *average* als für die Problemlösung tauglich beschrieben. Bei einem Zusammenschluß von Objekten bzw. Gruppen sind sie nicht auf eine echte Aktualisierung der Repräsentationen mit möglicherweise aufwendigen Neuberechnungen angewiesen, es wird letztlich auf einfache Weise ausschließlich mit den Werten der zu Beginn erzeugten Ähnlichkeitsmatrix gearbeitet. Dies ist insbesondere von Vorteil, wenn sich keine natürliche Gruppenrepräsentation anbietet.

Im vorliegenden Fall drängt sich jedoch eine solche geradezu auf: Da zwei Objekte durch ein Tupel von Mengen repräsentiert werden, liegt es nahe, die Fusion beider durch die komponentenweise Vereinigung dieser Mengen darzustellen.

Definition 3.5 *Die* **Ähnlichkeit zweier Mengen von Attributwerten** *X, Y des zu gruppierenden Attributs A einer Tabelle R einer relationalen Datenbank ist definiert als*

$$(1) \quad s(X,Y,A,R) := \sum_{k=1}^{n} w_k \cdot s_{B_k}(X,Y,A,R)$$

wobei $B_1 \ldots B_n$ vom Benutzer gewählte Attribute von R, $w_1 \ldots w_n$ deren zugeteilte Gewichte sind und $s_{B_k}(X,Y)$ als Ähnlichkeit von X und Y bzgl. des Attributs B_k wie folgt definiert ist:

$$(2) \quad s_{B_k}(X,Y,A,R) := \frac{|M_{B_k}(X,A,R) \cap M_{B_k}(Y,A,R)|}{|M_{B_k}(X,A,R) \cup M_{B_k}(Y,A,R)|}$$

mit

$$(3) \quad M_{B_k}(X,A,R) := \bigcup_{a_i \in X} \pi_{B_k}(\sigma_{A=a_i}(R))$$

Neben der resultierenden Kompaktheit — Definition 3.5 geht für einelementige X, Y in Definition 3.4 über, so daß letztlich nur ein Maß für alle drei Situationen Objekt–Objekt, Objekt–Gruppe, Gruppe–Gruppe verwendet wird — wirft diese Wahl auch praktischen Gewinn ab: Es wird so möglich, innerhalb der Gruppen nach Inklusionen zu suchen und diese als zusätzliche Strukturinformation dem Benutzer zugänglich zu machen (siehe Abschnitt 3.2.5).

Dafür, daß die Kosten für eine solche Repräsentation und die entsprechenden Berechnungen der Ähnlichkeitswerte im Rahmen bleiben, sorgt der in Abschnitt 3.2.4 beschriebene Zugriff auf die Daten und die Implementation, die die Ordnung der jeweiligen Mengen gewährleisten.

3.2.3 Nullwertbehandlung

Jedes Verfahren, das sich mit Attributwerten befaßt, muß eine (konsistente) Strategie finden, um mit fehlenden Werten möglichst angemessen umzugehen. Während fehlende Werte schon die herkömmliche Cluster-Analyse zu entsprechenden Überlegungen zwingen[15], stellt das pathologische $NULL$ in der Datenbanktheorie ein Kapitel für sich dar: Jeder fehlende Wert (Nullwert) ist eine Konstante, die sich von den anderen Nullwerten unterscheidet. Es gilt also $NULL \neq NULL$. Dies hat ganz offensichtlich Einfluß auf jede darauf aufbauende Logik. Ein Vorschlag, diesen „Schaden" aus Sicht des KDD möglichst elegant zu beheben, findet sich in [Bel95].

CLUST_DB ignoriert Nullwerte insofern, als daß sie nicht in die Mitspielermenge aufgenommen werden. Sie gehen also nicht in die Ähnlichkeitsberechnung mit ein.

3.2.4 Algorithmus

Der Wahl der hierarchischen bottom-up Cluster-Technik ist es zu verdanken, daß der Algorithmus leicht und kompakt zu beschreiben ist. Als entsprechend unproblematisch wird sich auch die Komplexitätsabschätzung bzgl. Platz- und Zeitbedarf erweisen, da die Verwendung von geordneten Mengen im Kern der Objektrepräsentation die Kosten der Ähnlichkeitsberechnungen klar ausweisen.

Die in Abschnitt 3.2.5 zu beschreibende Implementation weist an einigen Stellen Erweiterungen auf, die aber bzgl. der Komplexität der Datenstrukturen und des Ablaufs des Algorithmus keine Rolle spielen. Lediglich eine Erweiterung sei hier als Ausnahme vermerkt: Die Ähnlichkeitsmatrix wird mehrdimensional geführt, um Auswertungen bzgl. der einzelnen Quellattribute zu ermöglichen. Dies wäre für die Cluster-Analyse an sich nicht nötig gewesen.

Eingabe

- *Zielattribut* A — Attribut, dessen Werte gruppiert werden sollen.
- *Quellattribute* $B_1 \ldots B_n$ — für die Ähnlichkeitsbestimmung zu berücksichtigende Attribute.
- *Gewichte* $w_1 \ldots w_n$ — Gewichte, die für $B_1 \ldots B_n$ festlegen, wie stark diese in die Ähnlichkeitsbestimmung eingehen.

Ausgabe

- *binärer Baum* — Cluster-Hierarchie.

Datenstrukturen und ihr Platzbedarf

- $VALUES[1 \ldots i \ldots m_A] : (a_i)$
  Array, in dem die m_A verschiedenen Attributwerte von A vermerkt sind

[15] [Aha90] zeigte etwa, daß verschiedene Strategien zu teilweise stark unterschiedlichen Ergebnissen führen.

$\rightarrow \mathcal{O}(m_A)$.

- $PARTNER[1 \ldots i \ldots m_A] : (M_{B_1}(a_i), \ldots, M_{B_n}(a_i))$
  Array, in dem jeweils maximal $m_{B_{max}}$ Mitspieler bzgl. der Quellattribute aufgesammelt werden $\rightarrow \mathcal{O}(n \cdot m_A \cdot m_{B_{max}})$ — siehe Lemma 3.2.

- $CLUSTER : (a_i, left, right)$
  Binärer Baum für die Cluster-Hierarchie $\rightarrow \mathcal{O}(m_A)$ — ein vollständiger binärer Baum mit m_A Blättern hat $2m_A - 1$ Knoten (inkl. Blätter).

- $SIMMATRIX[1 \ldots i \ldots m_A][1 \ldots j \ldots m_A][0 \ldots k \ldots n] : (s_{B_k}(a_i, a_j))$
  Ähnlichkeitsmatrix bzgl. des Quellattributs B_k. An der Stelle $k = 0$ wird die Gesamtähnlichkeit gespeichert $\rightarrow \mathcal{O}(n \cdot m_A^2)$ — Platzbedarf von n $m_A \times m_A$-Matrizen.[16]

Algorithmus Clust_DB

(A) Initialisierung:

(1) Anzahl unterschiedlicher Objekte $m_A := |\pi_A(R)|$
$VALUES[1 \ldots m_A] := \pi_A(R)$
$PARTNER[1 \ldots i \ldots m_A] := M_{B_k}(VALUES[i])$ für alle Quellattribute $B_1 \ldots B_k \ldots B_n$

(2) Erzeuge m_A Blätter für die Attributwerte a_i im binären Baum $CLUSTER$.

(3) Fülle $SIMMATRIX[i][j][k]$ für $1 \leq i \leq j \leq m_A$ und $1 \leq k \leq n$ mit $s_{B_k}(a_i, a_j)$, den Ähnlichkeiten der Attributwerte a_i, a_j bzgl. des Attributs B_k. Speicher die jeweiligen Gesamtähnlichkeiten in den entsprechenden Feldern mit $k = 0$ ab.

(B) Gruppierung:

(1) Anzahl der Gruppen $g := m_A$.

(2) Entnehme $SIMMATRIX[i][j][0]$ das Paar (i, j), das die größte Ähnlichkeit aufweist.

(3) Erzeuge in $CLUSTER$ einen Vaterknoten für i, j und trage in ihm die Objektrepräsentation für das neue Objekt ein.

(4) Aktualisiere die Ähnlichkeitsmatrix bzgl. des neuen Objekts, das die Stelle von i einnimmt, und lösche das Objekt j.

(5) $g := g - 1$
Wenn $g > 1$, dann gehe zu (2).

(C) Ausgabe:

(1) Visualisiere die Hierarchie durch Ausgabe aller Knoten und Blätter von $CLUSTER$.

[16] Aufgrund der Symmetrie des Maßes und der Unwichtigkeit der Diagonaleinträge $s(i, i)$ werden nur obere Dreiecksmatrizen mit $m_A \cdot (m_A - 1)/2$ Einträgen benötigt, was aber an der Komplexität nichts ändert.

Korrektheit des Algorithmus

Der Algorithmus ist insofern korrekt, als daß offensichtlich in jedem Schritt die beiden ähnlichsten Objekte zusammengeführt werden. Dabei wird sowohl die Repräsentation für das neue Objekt erzeugt als auch die Ähnlichkeitsmatrix wie erforderlich aktualisiert.

Es ist sichergestellt, daß der Algorithmus in endlicher Zeit terminiert, da in jedem Schritt die anfangs m_A Gruppen um 1 verringert werden — es gibt stets ein ähnlichstes Paar — und der Algorithmus stoppt, wenn es nur noch eine Gruppe gibt.

Laufzeitabschätzung für den Algorithmus Clust_DB

Bei der Abschätzung wird zunächst der Aufwand der Datenbankzugriffe in (A1) vernachlässigt, was zu einer vorläufigen „internen" Laufzeitanalyse führt. Die Interaktion mit der Datenbank wird im Anschluß separat betrachtet.

(A) Initialisierung:

- (1) Abschätzung erfolgt später (siehe vorangegangene Bemerkung).
- (2) $\mathcal{O}(m_A)$ — trivial.
- (3) $\mathcal{O}(n \cdot m_A^2 \cdot m_{B_{max}})$ — Es sind $\mathcal{O}(m_A^2)$ Werte für jede der n Matrizen zu berechnen. Für einen Wert muß die Kardinalität der Vereinigung und des Schnitts zweier geordneter Mengen M_1, M_2 bestimmt werden. Dies kann in $\mathcal{O}(|M_1| + |M_2|)$ realisiert werden [AHU83]. Da die Kardinalität $|M_1|, |M_2|$ der beteiligten Mengen durch $m_{B_{max}}$ beschränkt ist, folgt insgesamt der angegebene Gesamtaufwand.

(B) Gruppierung:

- (1,5) $\mathcal{O}(m_A)$ Durchläufe von (2) – (4) — Bei jeder Fusion wird die Anzahl der Gruppen von anfangs m_A um 1 verringert.
- (2) $\mathcal{O}(m_A^2)$ — trivial.
- (3) $\mathcal{O}(n \cdot m_{B_{max}})$ — Das Herstellen eines Vaterknotens erfolgt in konstanter Zeit, während die Kosten der neuen Objektrepräsentation durch n-maliges Vereinigen geordneter Mengen bestimmt werden, deren Kardinalität höchstens $m_{B_{max}}$ beträgt.
- (4) $\mathcal{O}(n \cdot m_A \cdot m_{B_{max}})$ — Es sind $\mathcal{O}(m_A)$ Stellen der n Matrizen neu zu berechnen, da nur die beiden Zeilen $(i,\cdot),(j,\cdot)$ und die beiden Spalten $(\cdot,i),(\cdot,j)$ (teilweise — es handelt sich ja um obere Dreiecksmatrizen) aktualisiert werden müssen.

(C) Ausgabe:

- (1) $\mathcal{O}(m_A)$ — ein vollständiger binärer Baum mit m_A Blättern hat $2m_A - 1$ Knoten (inkl. Blätter).

Lemma 3.3 *Sei n die Anzahl der gewählten Quellattribute, N die Gesamtzahl der Tupel der entsprechenden Tabelle, m_A die Anzahl verschiedener vorkommender Attributwerte des Zielattributes A und $m_{B_{max}}$ das Maximum verschiedener vorkommender Werte der Quellattribute $B_1 \ldots B_n$, so ist die interne Laufzeit des Algorithmus* CLUST_DB *durch $\mathcal{O}(n \cdot m_A^2 \cdot m_{B_{max}} + m_A^3)$ beschränkt.*

Beweis Die Abschätzung folgt aus den einzelnen in der vorangegangenen Laufzeitabschätzung aufgeführten Positionen:

$$\mathcal{O}(\underbrace{n \cdot m_A^2 \cdot m_{B_{max}}}_{Initialisierung} + \underbrace{m_A \cdot (m_A^2 + 1 + n \cdot m_{B_{max}} + n \cdot m_A \cdot m_{B_{max}})}_{Gruppierung} + \underbrace{m_A}_{Ausgabe}) = \mathcal{O}(n \cdot m_A^2 \cdot m_{B_{max}} + m_A^3)$$

Eine Abschätzung durch $\mathcal{O}(n \cdot m_{max}^3)$ mit $m_{max} := max\{m_A, m_{B_{max}}\}$ wird als zu grob betrachtet, da sich $m_{B_{max}}$ etwa durch ein numerisches Quellattribut B_k um Größenordnungen von m_A unterscheiden kann. □

Während sich die Teile (B) und (C) des Algorithmus komplett im Speicher des Rechners abspielen und damit problemlos zu analysieren sind, sind bei der Initialisierung (A) Datenbankzugriffe nötig. Das wirft Fragen auf, die für die effektive Laufzeit wesentlich sein können:

(1) Wie sehen diese Anfragen an das Datenbankmagementsystem (DBMS) aus?

(2) Wie viele Zugriffe sind nötig?

(3) Was „kosten“ sie?

(4) Wie viele Attributwerte werden über das Netz bewegt?

Wenn möglich, ist es selbstverständlich anzustreben, (1) so zu wählen, daß (2),(3),(4) minimal sind. Es ist jedoch offensichtlich, daß man nicht unbedingt eine optimale Lösung erhält, wenn man nacheinander die einzelnen Punkte getrennt minimiert. So kann es sich lohnen, aufwendigere Anfragen zu stellen, um die Anzahl der gelieferten Tupel zu senken.

Bei der Frage nach den Kosten der Anfragen kann man lediglich referieren, was die Theorie besagt; das Verhalten des DBMS kann davon abweichen, es liegt letztlich außerhalb unserer Kontrolle.

Die Beantwortung der entscheidenden Frage (1) kann im eigentlichen Sinne auch als Teil der Implementation angesehen werden, wird aber ganz bewußt bereits in diesem Abschnitt erfolgen:

Der Initialisierungsschritt (A1) zur Bestimmung von m_A und zum Aufbau von $VALUES$ und $PARTNER$ wurde durch

`select distinct (` A, B_k `) from` R `order by` A

mit $k = 1 \ldots n$ realisiert.

Lemma 3.4 *Sei n die Anzahl der gewählten Quellattribute und N die Gesamtzahl der Tupel der entsprechenden Tabelle R, so kommt der Algorithmus* CLUST_DB *mit n Datenbankzugriffen aus, deren Gesamtaufwand durch $\mathcal{O}(n \cdot N \log N)$ beschränkt ist. Die Anzahl der selektierten Werte wird durch $\mathcal{O}(n \cdot m_A \cdot m_{B_{max}})$ beschrieben mit $m_A := |\pi_A(R)|$ und $m_{B_{max}} := max\{|\pi_{B_1}(R)|, \ldots, |\pi_{B_n}(R)|\}$.*

Beweis Zunächst ist festzuhalten, daß die beschriebenen Anfragen ausreichen, um die geforderten Informationen zu erhalten. $VALUES$ und $PARTNER$ werden offensichtlich bedient, dank der in Abschnitt 3.2.5 beschriebenen technischen Realisierung läßt sich darüber hinaus m_A durch einfaches „Mitzählen" der Tupel bestimmen. Die Anzahl der zu übermittelnden Werte wurde bereits in Lemma 3.2 bestimmt. Der mit $\mathcal{O}(N \log N)$ pro Anfrage angegebene Aufwand ergibt sich aus der Tatsache, daß jede dieser Anfragen N Paare (a, b) der Tabelle sortiert und der (berechtigten) Hoffnung, daß dies optimal im Sinne der Theorie[17] vom DBMS ausgeführt wird. □

Somit ergibt sich (ohne Beweis, da lediglich die beiden Teilergebnisse aus Lemma 3.3 und Lemma 3.4 zusammengeführt werden) das folgende Korollar, das die Laufzeit von CLUST_DB ausweist.

Korollar 3.1 *Sei n die Anzahl der gewählten Quellattribute, N die Gesamtzahl der Tupel der entsprechenden Tabelle, m_A die Anzahl verschiedener vorkommender Attributwerte des Zielattributes A und $m_{B_{max}}$ das Maximum verschiedener vorkommender Werte der Quellattribute $B_1 \ldots B_n$, so kommt der Algorithmus* CLUST_DB *mit n Datenbankzugriffen aus, wobei die Anzahl der selektierten Werte maximal $n \cdot m_A \cdot m_{B_{max}}$ beträgt. Die Laufzeit ist durch $\mathcal{O}(n \cdot N \log N + n \cdot m_A^2 \cdot m_{B_{max}} + m_A^3)$ beschränkt.*

3.2.5 Implementation

Nachdem schon im vorangegangenen Abschnitt ein Aspekt der Implementierung — die Auswahl der SQL-Anweisungen — besprochen wurde, werden im nun folgenden die Systemarchitektur in Bezug auf die Kopplung an eine relationale Datenbank sowie sämtliche Ausgaben des Programmes beschrieben. Die verschiedenen Nutzungsmöglichkeiten von CLUST_DB werden in Abschnitt 3.3 aufgezeigt.

Architektur

CLUST_DB wurde in der Programmiersprache C implementiert und konnte weitestgehend auf die in [Bro94] entworfene und beschriebene Schnittstelle zum DBMS Oracle 7 aufbauen. Die Schnittstelle setzt via *embedded SQL* die Anfragen von CLUST_DB ab und nimmt über ein Netzwerk mit TCP/IP-Protokoll Wert für Wert von der Datenbank in Empfang.

[17] Die angegebene Laufzeit läßt sich etwa durch Verwendung von HEAPSORT erreichen [AHU83].

Dieser inkrementelle Charakter des Transfers macht die Implementation an einigen Stellen leichter als es der relationale Zugang zunächst suggeriert. So können etwa beim Aufbau von *VALUE* und *PARTNER* (siehe Abschnitt 3.2.4) die über die Schnittstelle eintreffenden Werte bequem auf die Strukturen verteilt und gezählt werden.

Sowohl die Quellattribute als auch das Zielattribut dürfen einen beliebigen von Oracle 7 unterstützten Typ wie etwa *DATE, VARCHAR, NUMBER* aufweisen. Ausgenommen sind lediglich Attribute des Typs *RAW*, die unformatierte („rohe") Binärdaten beinhalten, die jeweils bis zu 2 GB groß sein können. Es steht dem Anwender aus technischer Sicht also auch der Einsatz von CLUST_DB bei numerischen Zielattributen offen.

Ausgabe

Nach diesen dem Benutzer verborgenen Details werden nun alle Ausgabemöglichkeiten der Implementation aufgeführt.

Wertehierarchie: Der binäre Baum, der die ermittelte Wertehierarchie trägt, wird einer Tiefensuche entsprechend ausgegeben. Dabei werden zu jedem Knoten die nötigen Informationen ausgegeben, die Aufschluß geben über seine Tiefe im Baum, den Zeitpunkt der Entstehung und die Ähnlichkeit, die zur Fusion der Söhne führte.

Zur Extraktion der in der Aufgabenstellung gesuchten Partitionen ist es nötig, an einer Ebene der Cluster-Hierarchie einen Schnitt anzusetzen. Dies wird vom System dahingehend unterstützt, daß die oben genannten Informationen dem Benutzer offengelegt werden. Die Wahl der Ebene bleibt aber letztlich dem Benutzer überlassen, es ist diesbzgl. keine Komponente entwickelt worden.

Ähnlichkeitsmatrizen und Varianz: Es wird für jedes Quellattribut B_k bestimmt, wir stark es den Gruppierungsprozeß beeinflußt hat. Als Maß wurde hierfür die Varianz der Ähnlichkeitswerte $s_{B_k}(a_i, a_j)$ gewählt:

$$\sigma_{s_{B_k}} = \sum_{1 \leq i \leq j \leq m_{B_k}} (s_{B_k}(a_i, a_j) - \overline{s}_{B_k})^2$$

Je kleiner dieser Wert ist, desto weniger hat das Attribut die Gruppierung bestimmt. Im Extremfall $\sigma_{s_{B_k}} = 0$ spielte B_k überhaupt keine Rolle. Neben der Varianz wird zusätzlich auch der Mittelwert $\overline{s}_{B_k}$ von $s_{B_k}(a_i, a_j)$ über alle Paare i, j ausgegeben. Auf Wunsch (Parameter) kann der Benutzer die Berechnung aller (Teil-)Ähnlichkeiten mitverfolgen.

Mitspielermengen: Für jeden Wert des Zielattributs wird ausgegeben, wieviele Werte sich in den verschiedenen Mitspielermengen befinden. Über einen Parameter kann der Benutzer neben dieser Information auch die explizite Ausgabe dieser Mengen herbeiführen.

Inklusionen: Bei jeder Berechnung eines Ähnlichkeitswertes wird überprüft, ob eine Inklusion dergestalt vorliegt, daß eine Mitspielermenge eines Objekts in der korrespondierenden Mitspielermenge eines anderen Objekts enthalten ist. Weist der Benutzer das Programm an, alle Ähnlichkeitswerte auszugeben, so wird eine Inklusion oder Gleichheit bzgl. einzelner Quellattribute angezeigt. Ist diese Inklusion für alle Komponenten gleichermaßen gültig, wird die Beziehung der beteiligten Objekte ausgegeben, sofern sie nicht trivial (eine der beiden Objekte weist nur leere Mengen auf) und die Übermenge durch die Repräsentation eines Wertes gegeben ist: Eine Inklusion der Form(en) $a_k(\cup a_l) \subseteq a_i \cup a_j$ wird unterdrückt, da sie im allgemeinen nicht sinnvoll interpretiert werden kann und wegen $X \subset Y \Rightarrow X \subset Y \cup Z$ zu einer Flut von Inklusionen führen könnte.

3.2.6 Wahl und Gewichtung der Quellattribute

In diesem Abschnitt wird es um die Wahl und die Gewichtung der Quellattribute $B_1 \dots B_n$ gehen.

Läßt man das in Abschnitt 3.1.1 Geschriebene über die *Principal Component Analysis* (PCA) Revue passieren, so kann man sich durchaus vorstellen, sie für die Wahl und Gewichtung der Quellattribute zu nutzen. Da dies jedoch kein Ansatz ist, der direkt in Zusammenhang mit dem von CLUST_DB verwendeten Maß gebracht werden kann, wurde diese Richtung nicht weiter verfolgt. Statt dessen wird an dieser Stelle der Bezug zum Konzept der *funktionalen Abhängigkeit* (im folgenden FA) hergestellt und für die Wahl und Gewichtung der Attribute genutzt.[18]

Es seien im folgenden stets A das Attribut aus Tabelle R, dessen Werte zu gruppieren sind und $B_1 \dots B_k$ paarweise (auch von A) verschiedene Attribute aus R.

Lemma 3.5 *Aus der Gültigkeit der FA $B_k \rightarrow A$ folgt, daß für alle $a_i, a_j \in \pi_A(R)$ mit $a_i \neq a_j$ gilt: $s_{B_k}(a_i, a_j) = 0$.*

Beweis Beweis durch Widerspruch: Sei also $B_k \rightarrow A$ und $s_{B_k}(a_i, a_j) \neq 0$ für mindestens zwei Attributwerte a_i, a_j von A mit $a_i \neq a_j$. Dann ergibt sich

$$\begin{aligned} s_{B_k}(a_i, a_j) \neq 0 \quad &\Rightarrow \quad M_{B_k}(a_i) \cap M_{B_k}(a_j) \neq \emptyset \\ &\Rightarrow \quad \exists v \in \pi_{B_k}(R) : v \in M_{B_k}(a_i) \wedge v \in M_{B_k}(a_j) \\ &\Rightarrow \quad (a_i, v), (a_j, v) \in \pi_{AB_k}(R) \end{aligned}$$

im Widerspruch zur Voraussetzung. □

Der Nutzen von Lemma 3.5 ist offensichtlich: Ist die Gültigkeit der FA $B_k \rightarrow A$ bekannt, so kann bei der Analyse der Attributwerte von A auf die Betrachtung von B_k verzichtet werden.

Sind die funktionalen Beziehungen $B_k \rightarrow B_l, B_l \rightarrow B_k$ — Kurzschreibweise $B_k \leftrightarrow B_l$ — bekannt, so läßt sich zeigen, daß bei entsprechender Gewichtung von B_k auf die Betrachtung von B_l verzichtet werden kann, ohne das Ergebnis zu verändern:

[18] Die Definition der funktionalen Abhängigkeiten läßt sich in Kapitel 2 finden.

Lemma 3.6 *Aus der Gültigkeit der FA $B_k \leftrightarrow B_l$ folgt, daß für alle $a_i, a_j \in \pi_A(R)$ gilt: $s_{B_k}(a_i, a_j) = s_{B_l}(a_i, a_j)$.*

Beweis Als Beweis wird plausibel gemacht, daß die Mitspielermengen bzgl. B_k und B_l strukturell identisch sind:

Durch die funktionale Abhängigkeit $B_l \leftrightarrow B_k$ läßt sich eine bijektive Abbildung $f : \pi_{B_k}(R) \rightarrow \pi_{B_l}(R)$ definieren, die jedem Wert aus $\pi_{B_k}(R)$ genau einen Wert aus $\pi_{B_l}(R)$ vermöge $f(b_k) = b_l :\Leftrightarrow (b_k, b_l) \in \pi_{B_k B_l}(R)$ zuordnet und umgekehrt. Auf diesem Weg kann in eindeutiger Weise durch Umbenennung der Elemente von $M_{B_k}(a_i)$ diese in die Menge $M_{B_l}(a_i)$ überführt werden. Gleiches gilt für a_j, so daß die berechneten Ähnlichkeitswerte, von der eventuell unterschiedlich gewählten Gewichtung abgesehen, übereinstimmen. □

Lemma 3.7 *Aus der Gültigkeit der FA $A \rightarrow B_k$ folgt, daß für alle $a \in \pi_A(R)$ gilt: $|M_{B_k}(a_i)| \leq 1$.*

Beweis Beweis durch Widerspruch: Sei $A \rightarrow B_k$ erfüllt und es sei $|M_{B_k}(a)| > 1$ für ein $a \in \pi_A(R)$, mit etwa $\{v, v'\} \subseteq M_{B_k}(a)$. Dann aber wäre $(a, v), (a, v') \in \pi_{AB_k}$ und damit $A \not\rightarrow B_k$, im Widerspruch zur Voraussetzung. □

Über den Fall $A \rightarrow B_k$ läßt sich somit bemerken, daß für die Ähnlichkeiten $s_{B_k}(a_i, a_j)$ gilt, daß sie gleich 0 oder gleich 1 sind, da die Mitspielermengen gemäß Lemma 3.7 jeweils einelementig oder leer sind.[19] Dies wird — abgesehen vom Fall $B_k \leftrightarrow A$ (siehe Lemma 3.5) — in der Regel bedeuten, daß B_k deutlich Einfluß auf die Gruppenbildung nehmen wird, insbesondere wenn $m_A \ll m_{B_k}$ gilt.

Neben einer Hilfestellung bzgl. der Wahl und der Gewichtung der Attribute bietet die Beziehung des Konzepts der FA zu den Ähnlichkeitswerten Ansätze für die Ausnutzung CLUST_DBs als KDD-Werkzeug zur Überprüfung der Datenqualität. Dieser Aspekt wird in Abschnitt 3.3.4 näher betrachtet, in dem gezeigt wird, inwiefern die Umkehrungen der in diesem Abschnitt bewiesenen Sachverhalte gelten.

Weitere für die Attributwahl und -gewichtung nützliche Informationen liefert die von CLUST_DB vor der eigentlichen Hierarchiebildung errechnete Varianz der Ähnlichkeitswerte bzgl. der betrachteten Attribute.[20] Da diese erst nach Berechnung der Ähnlichkeitsmatrix zur Verfügung stehen, können sie nicht wie bekannte FA *a priori*, sondern erst bei einem weiteren Lernlauf genutzt werden.

3.3 Einsatz von Clust_DB

In diesem Abschnitt werden die unterschiedlichen Einsatzmöglichkeiten von CLUST_DB beschrieben. Für all diese gilt gleichermaßen, daß der gewählte Ansatz nicht auf eine relationale Datenbank angewiesen ist. Es ist ein leichtes, CLUST_DB so umzuschreiben, daß etwa PROLOG-Fakten verarbeitet werden könnten.

[19] Die Mengen können leer sein, da eventuell in der Datenbank vorkommende Nullwerte nicht aufgenommen werden. Siehe hierzu auch Abschnitt 3.2.3.

[20] Siehe hierzu Abschnitt 3.2.5.

3.3.1 Typinduktion

Das Programm CLUST_DB ist in erster Linie die Umsetzung des in Abschnitt 3.2.4 beschriebenen Algorithmus — entwickelt, um gemäß der Aufgabenstellung in nominalen Attributen relationaler Datenbanken interessante Wertebereiche zu finden, was demnach auch den primären Einsatzzweck darstellt. Die Werte des vom Benutzer gewählten Attributs werden respektive des in Abschnitt 3.2.2 definierten Ähnlichkeitsmaßes agglomerativ gruppiert, um eine Hierarchie zu formen. Das Programm unterstützt den Benutzer bei der Wahl der Wertepartition dahingehend, daß alle Informationen, die Aufschluß darüber geben, wann und mit welcher Ähnlichkeit zwei Objekte zusammengeschlossen wurden, dem Benutzer offengelegt werden.

3.3.2 Datenanalyse

Auch wenn es nicht in erster Linie darum geht, ein Lernverfahren durch das (zusätzliche) Einführen von Typen zu unterstützen, so sind die von CLUST_DB ermittelten Gruppierungen im Sinne einer Datenanalyse dienlich. Zusammen mit den bzgl. der Mitspielermengen gefundenen Inklusionen können sie zu einem tieferen Verständnis der den Daten innewohnenden Struktur führen.

Gleiches gilt für das eigentliche „Zwischenergebnis“ — die berechnete Ähnlichkeitsmatrix: Die Kenntnis über die Ähnlichkeitswerte und deren Varianz stellt „Wissen per se“ dar, das aus der Datenbank gewonnen wurde. Man lernt, welche Attributwerte sich ähnlich in Bezug auf die anderen Attribute verhalten und welche Attribute besonders stark oder schwach trennen.

3.3.3 Ähnlichkeitsbestimmung

Neben der zuvor beschriebenen Ausnutzung der Ähnlichkeitsmatrix führt diese zu einer weiteren Einsatzmöglichkeit von CLUST_DB: Sie kann dazu herangezogen werden, um eine auf Tupeln basierende Cluster-Analyse mit den berechneten Ähnlichkeitswerten zu versorgen. Dabei sei in diesem Zusammenhang an Abschnitt 3.1.2 erinnert, in dem bemerkt wurde, daß die Bestimmung der Ähnlichkeiten bei nominalen Attributwerten zumeist wenig befriedigend verläuft. Als Einsatzbeispiel sei das Zusammenspiel mit dem Cluster-Verfahren KBG [Bis91] genannt, welches dem Benutzer die Möglichkeit bietet, zuvor bestimmte Ähnlichkeitswerte in den Gruppierungsprozeß einfließen zu lassen.

3.3.4 Funktionale Abhängigkeiten und Datenqualität

In Abschnitt 3.2.6 wurde der Zusammenhang zwischen dem gewählten Ähnlichkeitsmaß und den funktionalen Abhängigkeiten ausführlich diskutiert und der Nutzen für die Wahl und die Gewichtung der Quellattribute aufgezeigt. Dieser Bezug führt auch zu zwei eng miteinander verknüpften Nutzungsmöglichkeiten bzgl. der Entdeckung einstelliger funktionaler Abhängigkeiten und der Überwachung der Datenqualität: Die Gültigkeit einer funktionalen Abhängigkeit (FA) $A \rightarrow B$ kann mittels CLUST_DB unter bestimmten Umständen entdeckt bzw. verifiziert werden. Diese „Umstände“ haben mit möglicherweise in der Datenbank vorhandenen, aber von CLUST_DB vernachlässigten Nullwerten zu tun: Ist bekannt, daß sich kein Nullwert im Zielattribut A und Quellattribut B_k befindet, so kann auf zwei Wegen die Gültigkeit einer FA überprüft werden. Der erste Weg führt über die Ähnlichkeitsmatrix:

Lemma 3.8 *Gilt für alle $a_i, a_j \in \pi_A(R)$ mit $a_i \neq a_j$, daß $s_{B_k}(a_i, a_j) = 0$, so folgt daraus die Gültigkeit der FA $B_k \to A$.*

Beweis Beweis durch Widerspruch: Sei also $s_{B_k}(a_i, a_j) = 0$ für alle $a_i \neq a_j$ und $B_k \not\to A$, etwa mit $(a_i, v), (a_j, v) \in \pi_{AB_k}(R)$. Dann folgt

$$\begin{aligned}
(a_i, v), (a_j, v) \in \pi_{AB_k}(R) \quad &\Rightarrow \quad v \in M_{B_k}(a_i) \wedge v \in M_{B_k}(a_j) \\
&\Rightarrow \quad M_{B_k}(a_i) \cap M_{B_k}(a_j) \neq \emptyset \\
&\Rightarrow \quad \frac{|M_{B_k}(a_i) \cap M_{B_k}(a_j)|}{|M_{B_k}(a_i) \cup M_{B_k}(a_j)|} \neq 0 \\
&\Rightarrow \quad s_{B_k}(a_i, a_j) \neq 0
\end{aligned}$$

im Widerspruch zur Voraussetzung. □

Kommen Nullwerte vor, so kann es sein, daß wie gefordert alle $s_{B_k}(a_i, a_j)$ gleich 0 sind, aber dennoch keine gültige FA vorliegt: Die Definition 2.1 verbietet ebenso das Vorkommen eines Nullwertes in B_k wie auch das Vorkommen zweier Paare $(NULL, v), (NULL, v) \in \pi_{AB_k}(R)$ — da, wie bereits in Abschnitt 3.2.3 ausgeführt, $NULL \neq NULL$ gilt.

Die zweite Möglichkeit, eine FA nachzuweisen, bedient sich der Mitspielermengen:

Lemma 3.9 *Gilt für alle $a_i \in \pi_A(R)$ $|M_{B_k}(a_i)| \leq 1$, so folgt daraus die Gültigkeit der FA $A \to B_k$.*

Beweis Beweis durch Widerspruch: Sei $|M_{B_k}(a_i)| \leq 1$ für alle $a_i \in \pi_A(R)$ und $A \not\to B_k$. Sei etwa $(a_i, v), (a_i, v') \in \pi_{AB_k}$ das die FA verhindernde Paar. Dann aber wäre $|M_{B_k}(a_i)| \geq |\{v, v'\}| = 2$, im Widerspruch zur Voraussetzung. □

Analog zur Betrachtung der Ähnlichkeitsmatrix kann für den Fall, daß Nullwerte vorkommen, nicht die Gültigkeit der FA garantiert werden.

Weichen einige a_i, a_j von der Forderung $s_{B_k}(a_i, a_j) = 0$ bzw. $|M_{B_k}(a_i)| \leq 1$ ab, so deutet dies auf eine *annähernd korrekte* FA hin, die in Anbetracht der Tatsache, daß mit Datenfehlern (*noise*) in realen Datenbanken zu rechnen ist, ebenfalls bemerkenswert sein kann.

Während die in realen Datenbanken selten erfüllte Bedingung, daß keine Nullwerte vorliegen, den Einsatz für die Entdeckung einstelliger FA einschränkt, kann CLUST_DB doch leicht ohne weitere Voraussetzungen zur Falsifikation angenommener FA genutzt werden:

Korollar 3.2 *Gibt es $a_i, a_j \in \pi_A(R)$ mit $a_i \neq a_j$, so daß $s_{B_k}(a_i, a_j) \neq 0$, dann folgt daraus $B_k \not\to A$.*

Beweis Der Beweis ergibt sich unmittelbar aus Kontraposition von Lemma 3.5. □

Korollar 3.3 *Gibt es ein $a_i \in \pi_A(R)$, so daß $|M_{B_k}(a_i)| > 1$, dann folgt daraus $A \not\rightarrow B_k$.*

Beweis Der Beweis ergibt sich unmittelbar aus Kontraposition von Lemma 3.7. □

Verstöße gegen FA, die aus der Sicht des Benutzers jederzeit Gültigkeit haben sollten, könnten auf eine fehlerhafte Eingabe oder einen Datenfehler hinweisen. Sie können leicht bei der Inspektion der Mitspielermengen lokalisiert werden, so daß CLUST_DB auf diesem Weg einen Beitrag leistet, die Datenqualität zu überprüfen.

3.3.5 Tupel-Cluster-Analyse

CLUST_DB wurde bislang als Verfahren beschrieben, das entgegen dem klassischen Vorgehen nicht Tupel gruppiert, sondern einzelne Attributwerte eines ausgezeichneten Attributs. Durch spezielle Wahl des Zielattributs bzw. durch einen kleinen Kunstgriff kann CLUST_DB auch zur Cluster-Analyse von unklassifizierten Beobachtungen herangezogen werden, die durch ein Tupel beschrieben werden:

Wird ein Attribut mit paarweise voneinander verschiedenen Werten als Zielattribut bestimmt bzw. bewußt zu diesem Zweck eingeführt und alle anderen als Quellattribute gewählt, so entspricht die Mitspielerrepräsentation der im Abschnitt 3.2.2 beschriebenen Tupelrepräsentation, da alle Mitspielermengen — von Nullwerten abgesehen — einelementig sind. Auf diesem Wege kann eine Cluster-Analyse erzwungen werden, die über die „Kennzeichen“ der Tupel zur Gruppierung dieser führt.

Dieses Vorgehen ist mit zwei Einschränkungen behaftet: Nicht umsonst wurde die Tupelrepräsentation verworfen, da alle N Tupel in den Speicher geladen werden müssen. Dies wird nur für kleinere Datenbanken durchführbar sein, da insbesondere auch die Aufstellung und Verwaltung der Ähnlichkeitsmatrizen dann für n Attribute mit einem Platzbedarf und Laufzeitverhalten von $\mathcal{O}(n \cdot N^3)$ zu Buche schlagen.[21]

Darüber hinaus stellt sich die Ähnlichkeitsberechnung auf Objektebene für diesen Fall trivial dar: Unabhängig vom Datentyp der Quellattribute werden die Einträge lediglich komponentenweise auf symbolische Gleichheit getestet.

Während eine Erweiterung CLUST_DBs in Bezug auf den letzten Punkt durchaus denkbar ist und in Abschnitt 3.6.3 aufgegriffen wird, so stellt erstgenanntes ein allgemeines Problem dar, welches vielen Cluster-Verfahren eine praktikable Umsetzung für Datenbanken verwehrt.

3.3.6 Dimensionsreduktion

Analog zu dem beschriebenen „Kunstgriff“, CLUST_DB zu einer Tupel-Cluster-Analyse zu bewegen, führt die Wahl eines vorhandenen Klassifikationsattributs als Zielattribut zu einer sehr interessanten praktischen Anwendung: CLUST_DB

[21] Dies folgt unmittelbar aus Lemma 3.1 mit $m_A = N$.

kann in diesem Fall dazu genutzt werden, die für einen Begriffslerner irrelevanten Attribute zu bestimmen, um diese als Vorverarbeitungsschritt vor dem eigentlichen Lernen aus der Datenmenge zu entfernen.

So verblüffend dies auch erscheinen mag, so einfach kann es vor Augen geführt werden: Der Schlüssel hierzu ist in diesem Fall nicht die von CLUST_DB ermittelte Gesamtähnlichkeit der unterschiedlichen Klassen, sondern vielmehr die ebenfalls gelieferte Auswertung der Ähnlichkeiten bzgl. der einzelnen Quellattribute. Je kleiner der Mittelwert $\overline{s}_{B_k}$ von $s_{B_k}(a_i, a_j)$ über alle Klassen a_i, a_j ist, um so mehr versteht es dieses Attribut, die unterschiedlichen Klassen zu separieren. Sind im Extremfall alle $s_{B_k}(a_i, a_j) = 0$, so genügt allein B_k, um die Klassen zu trennen. Sind im anderen Extrem alle $s_{B_k}(a_i, a_j) = 1$, so ist es sehr wahrscheinlich, daß B_k etwa beim Lernen eines Entscheidungsbaumes eine untergeordnete Rolle spielt, da jeder Wert aus B_k zumindest einmal jeder Klasse zugeordnet wurde. Auch wenn dies eben nur „wahrscheinlich“ ist und keineswegs bedeutet, daß auf solche Attribute in jedem Fall ohne Verlust an Genauigkeit verzichtet werden kann, bietet es sich dennoch an, dieses Attribut und deren Werte aus den Daten zu entfernen. Ob Versuchsergebnisse die Tauglichkeit von CLUST_DB als Hilfsmittel zur Dimensionsreduktion belegen, wird im folgenden Abschnitt 3.4.2 untersucht.

3.4 Versuchsergebnisse

Gegenstand der Betrachtung sollen im folgenden Experimente mit CLUST_DB sein. Diese sollen zum einen Aufschluß darüber geben, ob der im Abschnitt 3.3.6 beschriebene Einsatz zur Datenreduktion für ein Begriffslernverfahren sich in der Praxis bewährt.

Bevor dieser Punkt untersucht wird, soll zunächst jedoch eine Versuchsreihe das Laufzeitverhalten von CLUST_DB verdeutlichen.

3.4.1 Laufzeiten

Zur Untersuchung des praktischen Laufzeitverhaltens wurden Tabellen der Firma SYLLOGIC verwendet, denen an dieser Stelle für die Bereitstellung gedankt sei.

In Tabelle 3.2 sind die für die Laufzeit von CLUST_DB bestimmenden Parameter wie Anzahl der vorkommenden verschiedenen Werte des Zielattributs (m_A), die Anzahl der Quellattribute (n) und die Anzahl der maximal in einem der Quellattribute vorkommenden verschiedenen Werte ($m_{B_{max}}$) angegeben. Die erzielten Zeiten[22] für die jeweiligen Läufe (gesamt) werden aufgeschlüsselt für das Lesen der Tupel (fetch), den initialen Aufbau der Ähnlichkeitsmatrizen (init) und schließlich das eigentliche Gruppieren der Objekte mit der zugehörigen Aktualisierung der Matrizen (group).

Tabelle 3.2 verdeutlicht, daß CLUST_DB für den Fall sehr schnell die Wertehierarchie aufbaut, wenn die Anzahl der Objekte verhältnismäßig klein (in der

[22] SUN Sparc Station 20 (DB-server: Sparc 10)

m_A	n	$m_{B_{max}}$	gesamt	fetch	init	group
5	3	1521	26”	24”	1”	1”
5	6	1521	51”	47”	2”	2”
23	3	1521	14”	11”	1”	2”
23	6	1521	28”	21”	1”	6”
90	3	1521	56”	16”	9”	31”
90	6	1521	2’ 8”	32”	19”	1’ 17”
90	4	23	6”	4”	1”	1”
1521	3	90	$\approx 32h$	55”	6’ 5”	$\approx 32h$

Tabelle 3.2: Laufzeiten für CLUST_DB

Größenordnung von 100) ist. Dies entspricht der zugrundegelegten Annahme, daß mit wenigen verschiedenen nominalen Attributwerten zu rechnen ist.

Man erkennt, daß der Zeitaufwand für das Lesen der Tupel linear mit der Anzahl der verwendeten Quellattribute steigt und daß die betrachteten Werte für $m_{B_{max}}$ kein Problem darstellen.

Wird allerdings die Arbeitshypothese dahingehend verletzt, daß viele Objekte (in der Größenordnung von 1000) zu bearbeiten sind, steigt der Zeitaufwand enorm an, auch wenn nur wenige Quellattribute gewählt wurden und sich ein geringer Wert für $m_{B_{max}}$ ergab.

3.4.2 Dimensionsreduktion

Im Abschnitt 3.3.6 wurde ausgeführt, daß die Auswertung der Ähnlichkeitswerte für die einzelnen Quellattribute zur Datenreduktion genutzt werden kann. Um diesen Ansatz auf seine Tauglichkeit hin zu überprüfen, wurde der Begriffslerner C4.5 [Qui93] auf die mit CLUST_DB reduzierten Datensätzen gestartet und das gelernte Ergebnis mit dem zuvor ohne Reduktion erzielten verglichen. Als Lernmaterial dienten hierzu vier Datensätze der bekannten Sammlung der University of California, Irvine:

Glass Identification Database: Es gilt hier, anhand 9 numerischer Attribute 7 verschiedene Typen von Glas zu unterscheiden. Insgesamt sind 214 Beispiele gespeichert. Diese Untersuchung ist durch die Kriminologie motiviert, bei der das am Tatort zurückgebliebene Glas als Beweis genutzt werden kann — sofern es korrekt identifiziert wird!

Hepatitis: Daten zu 6 numerischen und 13 binären Attributen wurden von 155 an Hepatitis erkrankten oder verstorbenen Patienten zusammengetragen, um zu lernen, wann eine Erkrankung tödlich verläuft.

Iris: Der von Fisher in seinem klassischen Artikel über die Diskriminant-Analyse [Fis36] benutzte Datensatz umfaßt 150 Beispiele der drei unterschiedlichen Iris-Spezien, deren charakteristische Eigenschaften zu lernen sind. Jedes dieser Beispiele wird durch 4 unterschiedliche numerische Attribute beschrieben.

Labor-negotiations: Anhand von 16 Attributen (davon 7 numerisch, 5 nominal und 3 binär) wurden 40 Vertragsverhandlungen erfaßt. Lernziel ist die Bestimmung der Rahmenbedingungen, die einen Vertrag (un)annehmbar machen.

Da CLUST_DB die vorhandenen numerischen Attribute nicht adäquat behandelt, wurden diese zuvor durch NUM_INT (Gap (*Hepatitis*), Discontinuity (*Labor-neg.*)), EQUAL-FREQUENCY (*Glass*) und EQUAL-WIDTH (*Iris*) diskretisiert.

Nachdem CLUST_DB für jeden Datensatz mit dem jeweiligen Klassifikationsattribut als Zielattribut und den verbleibenden als Quellattributen die Ähnlichkeitsmatrizen bestimmt hatte, wurden anhand dieser gemäß 3.3.6 diejenigen Attribute aus den Datensätzen entfernt, die die größten $\overline{s}_{B_k}$ aufwiesen.[23] Auf die so reduzierten Daten wurde C4.5 angesetzt, um die erzielten Ergebnisse mit denen zu vergleichen, die auf den lediglich diskretisierten Daten erzielt wurden.

Die in Tabelle 3.3 angegebenen Reduktion bezieht sich ausschließlich auf die Attributanzahl und ist insofern als Untergrenze einer allgemeineren Datenreduktion zu verstehen, als daß nicht überprüft worden ist, ob durch den Wegfall der Attribute einige Tupel mehrfach vorkamen und im Sinne einer weiteren Reduktion vernachlässigbar gewesen wären.

Datensatz	gewählte Reduktion	Anstieg der Fehlerrate
Glass	45% (9 → 5)	5.7 (9.3% → 15.0%)
Iris	50% (4 → 2)	0.6 (4.7% → 5.3%)
Labor-neg.	44% (16 → 9)	0.0 (2.5% → 2.5%)
Hepatitis	63% (19 → 7)	6.4 (6.5% → 12.9%)

Tabelle 3.3: Versuchsergebnisse für CLUST_DB als Hilfsmittel zur Dimensionsreduktion. In der Spalte *gewählte Datenreduktion* ist in Klammern angegeben, wieviele Attribute vor und nach der Dimensionsreduktion vorhanden waren. In der Spalte *Fehlerrate* sind in Klammern die Ergebnisse der Lernläufe ohne / mit vorangegangener Dimensionsreduktion angegeben.

Tabelle 3.3 verdeutlicht, daß für die Datensätze *Iris* und *Labor-neg.* hervorragende Ergebnisse erzielt werden konnten. So stieg die Fehlerrate bei *Labor-neg.* überhaupt nicht und bei *Iris* nur sehr wenig im Vergleich zur erzielten Datenreduktion. Bei letztgenannter Domäne, die eine vergleichsweise leichte Lernaufgabe darstellt, da die einzelnen Spezies mit sehr geringer Überlappung linear separierbar sind, fiel die Entscheidung besonders leicht, anhand der im Anhang A.2 angegebenen Ähnlichkeitswerte zu bestimmen, wieviele (welche) Attribute sinnvoll zu vernachlässigen sind.

Bei den anderen beiden Domänen wurde die erzielte Reduktion mit einem spürbaren Anstieg der Fehlerrate erkauft. Bei *Glass* ist dies sogar als positiv zu werten, da die von CLUST_DB ermittelten Ähnlichkeitswerte recht deutlich

[23] Die ermittelten Ähnlichkeitswerte und die Angabe, welche Attribute entfernt wurden, können dem Anhang entnommen werden.

anzeigen, daß alle Attribute (annähernd gleich) wichtig für die korrekte Klassifikation sind.[24]

Anders liegt der Fall bei *Hepatitis*. Hier überrascht das schlechte Abschneiden zunächst, da die im Anhang A.3 wiedergegebenen Werte deutlich für eine effektive Dimensionsreduktion zu sprechen scheinen. Zwei Dinge sind letztlich für die erhöhte Fehlerrate bei *Hepatitis* verantwortlich zu machen:

1. CLUST_DB beachtet aufgrund der in Abschnitt 3.2.2 gewählten Mengenrepräsentation der Objekte nicht die Häufigkeit, mit denen die Werte gelesen wurden. Dies ist für den Datensatz *Hepatitis* unvorteilhaft, da 13 binäre Attribute vorkommen. Insbesondere in dem Fall, daß einige Beispiele falsch klassifiziert wurden (*noise*), geben die berechneten Ähnlichkeitswerte nicht korrekt Aufschluß über die tatsächliche Trennschärfe bzgl. der Klassen.

2. Der in Abschnitt 3.3.6 skizzierte Ansatz zur Dimensionsreduktion ist heuristisch in dem Sinne, daß der Erfolg dieses Vorgehens nicht garantiert werden kann. Insbesondere dann, wenn eine schwierige Lernaufgabe vorliegt, die die Induktion eines großen Entscheidungsbaumes nötig macht, wird (verstärkt) auch auf solche Attribute zurückgegriffen, die die beschriebene Heuristik als nicht relevant einstuft. In der Tat war der für *Hepatitis* ohne Datenreduktion bestimmte Entscheidungsbaum (49) deutlich größer als der für *Iris* (16) und *Labor-neg.* (21).

3.5 Vergleich mit anderen Verfahren

In diesem Abschnitt werden kurz Verfahren beschrieben, von denen jedes auf seine Weise mit CLUST_DB in Bezug gebracht werden kann. Dabei wird als Beispiel eines auf Tupeln basierenden Cluster-Verfahrens COBWEB [Fis87] vorgestellt werden. Der Betrachtung des CLUST_DB näher stehenden Verfahrens KBG [Bis91] folgt eine Abgrenzung vom Gebiet der distanzbasierten Lernverfahren.

3.5.1 COBWEB

Bei COBWEB [Fis87] handelt es sich um ein Cluster-Verfahren, das aus einer Sequenz von unklassifizierten Beobachtungen, die jeweils durch ein Tupel von Attributwerten beschrieben werden, top-down eine Hierarchie erzeugt. Anders als Verfahren der Cluster-Analyse, die ihre Wurzeln in der Statistik haben, entspringt COBWEB dem Gebiet des maschinellen Lernens. Dort werden Verfahren, die Beobachtungen anhand ihrer Beschreibung in Klassen einteilen, unter dem Begriff *conceptual clustering* zusammengefaßt. Diese sprachliche Abgrenzung (der Begriff geht auf [MS83] zurück) soll verdeutlichen, daß es für diese Verfahren im Gegensatz zur Cluster-Analyse wesentlich ist, Beschreibungen der Gruppierungen zu finden, die für den Menschen verständlich sind. So wird anders als bei CLUST_DB, das eine Gruppierung lediglich durch das Auflisten seiner Elemente beschreibt, bei COBWEB die von der Spitze zunehmend spezifischen

[24] Siehe hierzu Anhang A.1.

Gruppierungen durch einen „repräsentativen“ Wert dargestellt — begleitet von der Auskunft, wie stark dieser Wert innerhalb der Gruppe variiert.[25]

Im Lichte der Ausführungen in Abschnitt 3.1.3 stellen sich bei den hierarchischen top-down Verfahren die Fragen nach der Bewertung einer Gruppierung (*objective function*) und der Heuristik (*control strategy*), die steuert, welche möglichen Zuordnungen getestet werden: Die Güte eines Clusters wird bei COBWEB durch deren „Vorhersagekraft“ bestimmt, die als

$$\Pi(C) = \frac{1}{I} \sum_{i=1}^{I} \Pi(A_i, C);$$

mit I gleich der Anzahl der Attribute definiert ist. Dabei ist V die Vorhersagekraft eines Attributs A_i vermöge

$$\Pi(A_i, C) = \sum_{i=1}^{J(i)} P(A_i = V_{ij}|C)^2$$

mit $J(i)$ gleich der Anzahl möglicher Werte für A_i. Die Qualität einer Partitionierung eines Clusters C in $\{C_1, \dots, C_k\}$ ergibt sich — ähnlich wie etwa bei ID3 [Qui86] — durch die gewichtete Zunahme der Vorhersage $1/K \sum_{k=1}^{K} P(C_k) \cdot (\Pi(C_k) - \Pi(C))$.

Hinter diesen Definitionen steht das Bestreben, Gruppierungen dergestalt zu finden, daß allein aus der Kenntnis der Zuordnung eines Objektes zu einer Gruppe eine möglichst gute Vorhersage über die tatsächlichen Attributwerte des Objektes erzielt wird.

Das Problem, daß es exponentiell viele Möglichkeiten gibt, unterschiedliche Gruppierungen zu bilden, wird insofern umgangen, als daß die Beobachtungen nacheinander (*inkrementell*) verarbeitet werden.[26] Diese einfache Heuristik läßt die effektive Bearbeitung auch größerer Datenmengen zu, erkauft sich dies aber zu dem Preis, daß die gebildete Hierarchie abhängig von der gewählten Reihenfolge der Objekte ist.

Dabei durchläuft das zu betrachtende Objekt top-down die bereits gebildete Hierarchie gemäß seiner Attributwerte und verändert diese ggf. durch die beiden Operatoren *merge*, *split* und *create*. Erstgenannter wird aktiviert, wenn das neue Objekt anzeigt, daß (nun) ein Zusammenführen zweier Söhne besser im Sinne der Vorhersagekraft ist. Analog dazu kann ein Objekt eine Aufspaltung bewirken, die einen Cluster in speziellere Teilgruppierungen zerfallen läßt. Mit *create* schließlich können neue Knoten in der Hierarchie erzeugt werden.

Auch wenn Gemeinsamkeiten bei CLUST_DB und COBWEB zu entdecken sind — beides sind hierarchische Cluster-Verfahren — so trennt die unterschiedliche Objektrepräsentation zu stark, um sie als ähnlich zu bezeichnen. Wie bei CLUST_DB so ist auch COBWEB untrennbar mit der verwendeten Repräsentation und dem darauf aufbauenden Ähnlichkeitsmaß verbunden: Im Gegensatz zu CLUST_DB bildet COBWEB Gruppierungen nicht innerhalb einzelner Attribute, sondern über die Tupel.

[25] Siehe diesbzgl. auch die Diskussion in Abschnitt 3.6.2.

[26] Inkrementelle Verfahren des conceptual clustering werden auch unter *concept formation* geführt.

Aus diesem Grunde wird auch darauf verzichtet, weitere Cluster-Verfahren wie etwa UNIMEM [Leb87] und AUTOCLASS [CS96] zu besprechen, die auf der gleichen Repräsentation beruhen.

3.5.2 KBG

Ein Verfahren, welches das verbreitete Schema der Attribut-Werte-Repräsentation aufbricht, ist das im Laufe dieser Arbeit bereits erwähnte KBG [Bis91]: Die von KBG (*KBG: Knowledge Based Generalizer*) verwendete Repräsentation entspricht im wesentlichen der Prädikatenlogik erster Stufe ohne negative Literale und Funktionssymbole. Erweitert wurde diese um die Behandlung numerischer Werte und um Prädikate, die prozeduale Auswertungen ermöglichen.

Eine Beobachtung T ist bei KBG von der Form

$$T := p(X_1, \ldots, X_n, V_1, \ldots V_m)$$

wobei $X_1, \ldots, X_n$ *Objekte* und $V_1, \ldots V_m$ *Werte* sind, deren getrennte Behandlung wesentlich für die Ähnlichkeitsbestimmung ist.

Zur Berechnung der Ähnlichkeit $SIM(X_r, X_r')$ zweier Objekte X_r, X_r' werden zunächst alle Beobachtungen T, T' ausgewertet, in denen X_r, X_r' an einer gemeinsamen Stelle r eines Prädikats p vorkommen:

$$T - SIM(T, T') := \Big(\frac{1}{n}\sum_{i=1}^{n} s_{Objekt}(X_i, X_i') \cdot \frac{1}{m}\sum_{j=1}^{m} s_{Wert}(V_j, V_j')\Big) \cdot w(p)$$

mit dem Gewicht des Prädikats $w(p)$ und

$$s_{Objekt}(X_i, X_i') = SIM(X_i, X_i') \text{ falls } i \neq r, \text{ 1 sonst }^{27}$$

$s_{Wert}(V_j, V_j')$ ist die Ähnlichkeit zweier Werte in Abhängigkeit der Untersorte (z. B. nominal, numerisch) und kann vom Benutzer entweder durch Angabe einer entsprechenden Funktion oder einzelner Ähnlichkeitswerte bestimmt werden.

Sei k_{co} die Anzahl aller Beobachtungen T_i, T_i', in denen X_r, X_r' an einer gemeinsamen Stelle r eines Prädikats p vorkommen und $occ(X)$ die Menge der Prädikate, in denen das Objekt X vertreten ist, dann ist die Ähnlichkeit der beiden Objekte X_r, X_r' gegeben durch

$$SIM(X_r, X_r') = \frac{\sum_{i=0}^{k_{co}} T - SIM(T_i, T_i')}{max(\sum_{p \in occ(X_r)} w(p), \sum_{p \in occ(X_r')} w(p))}$$

Die Ähnlichkeit der Beobachtungen wird schließlich aus dem Mittelwert der Ähnlichkeiten der beteiligten Objekte ermittelt.

Sind alle Ähnlichkeitswerte berechnet, beginnt ein bottom-up Gruppierungsprozeß, in dessen Verlauf anders als bei CLUST_DB Beobachtungen generalisiert

[27] Dadurch, daß so die Ähnlichkeit der anderen beteiligten Objekte berücksichtigt wird, kommt es für den Fall, daß mehrere Objekte in einem Prädikat vorkommen, zu einem Gleichungssystem, das von KBG gelöst werden muß, um die gesuchte Ähnlichkeit der Objekte X_i, X_i' zu erhalten. Es wird in [Bis91] gezeigt, daß dies stets in eindeutiger Weise möglich ist.

werden und kein Hierarchiebaum, sondern ein azyklischer gerichteter Graph bestimmt wird.

Beschränkt man sich zunächst auf die Bestimmung der Ähnlichkeitswerte von Objekten sind Gemeinsamkeiten zu CLUST_DB erkennbar: Werden zwei Tupel $(a, b_1 \ldots, b_n), (a', b'_1, \ldots, b'_n)$ einer Tabelle einer relationalen Datenbank für KBG als zwei Beobachtungen $p(a, b_1, \ldots, b_n)$ bzw. $p(a', b'_1, \ldots, b'_n)$ mit a, a' als Objekte und $b_1^{(')}, \ldots, b_n^{(')}$ als Werte repräsentiert, so kann KBG die Ähnlichkeit für a, a' in etwa so bestimmen, wie CLUST_DB dies bei der Verwendung der Tupelrepräsentation (siehe Abschnitt 3.2.2) getan hätte.

Da KBG als Standardtyp für Werte auch geordnete Mengen anbietet, ist sogar die der von CLUST_DB verwendeten Mitspielerrepräsentation entsprechende Form

$$p(a, \{b_{1,1}, \ldots, b_{m_{B_1},1}\}, \ldots, \{b_{1,n}, \ldots, b_{m_{B_n},n}\})$$

und das darauf basierende Maß zu erzielen.[28] Dabei würde KBG beim Generalisieren zweier Beobachtungen automatisch die neue Repräsentation durch Vereinigung der beteiligten Mengen realisieren. Daß KBG an sich die Beobachtungen und nicht die Objekte gruppiert, spielt an dieser Stelle dann keine Rolle mehr: Es gibt keine zwei Beobachtungen mit dem gleichen Objekt, so daß Beobachtung und Objekt eindeutig einander zugeordnet werden können. Außerdem ist die Ähnlichkeit zweier Beobachtungen dann eben genau die Hälfte des Ähnlichkeitswertes der Objekte (es kommen ja nur diese beiden in den betrachteten Beobachtungen vor), so daß letztendlich die ähnlichsten Objekte gruppiert werden. Würde also eine Schnittstelle implementiert, die die beschriebene Übersetzung der Tupel für KBG realisiert, so dürfte KBG in Bezug auf die Objekthierarchie zu ähnlichen Ergebnissen kommen wie CLUST_DB.

3.5.3 Distanzbasierte Lernverfahren

Unter dem Begriff *Distance Based Learning Algorithms* werden Verfahren geführt, die unklassifizierte *Anfragen* anhand der Distanz zu den in einer Trainingsmenge gespeicherten klassifizierten Beispielen oder deren vom Verfahren bestimmten Generalisierungen den unterschiedlichen Klassen zuordnen.

Im einfachsten Fall des *nearest neighbour*-Ansatzes entscheidet das Trainingsbeispiel über die Klassenzugehörigkeit der Anfrage, das bzgl. eines Distanzmaßes sich am nächsten zu dieser befindet. Eine naheliegende Erweiterung des Konzeptes stellen die kNN-Verfahren (kNN: *k-nearest-neighbour*) dar: Es werden für eine zu klassifizierende Anfrage q zunächst die k nächsten Trainingsbeispiele bestimmt, um diese per Mehrheitsentscheid (gleichberechtigt oder mit der jeweiligen Nähe zu q gewichtet) über die Klassenzugehörigkeit von q abstimmen zu lassen.

Verfahren der NGE-Theorie (NGE: *Nested Generalized Exemplars* [Sal91]) betrachten nicht nur einzelne Trainingsbeispiele, sondern erzeugen vielmehr aus diesen Generalisierungen, die die Form von achsenparallelen, mehrdimensionalen Rechtecken (*hyperrectangles*) aufweisen, die ineinander verschachtelt (*ne-*

[28] Die unterschiedliche Gewichtung der Quellattribute kann nicht nachgebildet werden, da das in der Formel angegebene $w(p)$ sich nur auf unterschiedliche Prädikate bezieht.

sted) sein können. Die Anfragen werden dabei dann der Klasse der nächsten umschließenden Generalisierung zugeordnet.

Offensichtlich ist die Frage nach dem verwendeten Distanzmaß auch hier von ganz besonderem Interesse. Im Vergleich zu CLUST_DB und den anderen in diesem Abschnitt vorgestellten Ansätzen ist die Aufgabenstellung jedoch eine ganz andere, weswegen hier nicht weiter auf Details eingegangen wird. Insbesondere zum Vergleich zwischen kNN und NGE sei [WD95] sehr empfohlen.

3.6 Diskussion

Dieser Abschnitt bildet den Schluß des dritten Teils der Arbeit, in dem es im Rahmen der Aufgabenstellung, interessante Wertebereiche in Attributen einer relationalen Datenbank zu finden, um die Behandlung nominaler Attribute ging.

Ehe sich die Arbeit den numerischen Attributen zuwendet, wird in diesem Abschnitt auf das dritte Kapitel zurückgeblickt, um die wesentlichen Ergebnisse zusammenzufassen und diese kritisch zu bewerten. Schließlich werden in Abschnitt 3.6.3 mögliche Verbesserungen diskutiert, die (noch) nicht umgesetzt wurden.

3.6.1 Zusammenfassung

Nachdem im Abschnitt 1.1 die getrennte Bearbeitung von nominalen und numerischen Attributen motiviert wurde, befaßte sich das dritte Kapitel mit der Aufgabe, in nominalen Attributen relationaler Datenbanken nach Gruppierungen zu suchen, die einem Lernverfahren sowohl durch die erzielte Datenreduktion als auch als zusätzliches Vokabular (*predicate invention*) dienen können.

Dem Ansatz folgend, die Aufgabe über die Cluster-Analyse anzugehen, wurden im Abschnitt 3.1 die wesentlichen Grundzüge unterschiedlicher Verfahren vorgestellt. Es mußte schnell bemerkt werden, daß die vorliegende Ausgangssituation, in der die zu gruppierenden Objekte durch einzelne Attributwerte gegeben sind, zwar wesentlich ist für die Durchführbarkeit einer Cluster-Analyse, doch gerade in Bezug auf das zu verwendende Ähnlichkeitsmaß — den Kern eines jeden Cluster-Ansatzes — entsprechende Überlegung erfordert: Keines der bekannten Konzepte war in der Lage, als Maß zu dienen, da sie darauf beruhen, daß die Objekte durch Tupel von Attributwerten beschrieben werden. So wurde schließlich ein eigenes Ähnlichkeitsmaß definiert, das zusammen mit der gewählten hierarchischen bottom-up Cluster-Technik das „Innenleben" des entworfenen und in Abschnitt 3.2 vorgestellten CLUST_DB ausmacht. Das im Abschnitt 3.2.2 definierte Ähnlichkeitsmaß macht — vereinfacht ausgedrückt — die Ähnlichkeit zweier Attributwerte a_i, a_j des zu gruppierenden Attributs an der Übereinstimmung der Wertemengen fest, die in Kombination mit a_i bzw. a_j in der Datenbank zu finden sind.

Die Beschäftigung in Abschnitt 3.2.6 mit der Frage, welche Attribute sich an der Bildung der eben beschriebenen nötigen Wertemengen als Quellattribute beteiligen können bzw. sollten, führte zur Betrachtung des Konzepts der funktionalen Abhängigkeiten, deren sinnvolle Ausnutzung durch einige Lemmata aufgezeigt wurde. Gewissermaßen als Nebenprodukt fielen dabei im Abschnitt

3.3.4 Korollare ab, die den Einsatz von CLUST_DB um die Entdeckung und Validierung von funktionalen Abhängigkeiten und, damit verbunden, der Überprüfung der Datenqualität erweitern.

Das Ähnlichkeitsmaß und der zusätzliche Aufwand bei der Implementierung von CLUST_DB bzgl. mehrdimensionaler Matrizen sorgten letztlich für einen weiteren Punkt: Wird CLUST_DB darauf angesetzt, Gruppierungen innerhalb eines Klassifikationsattributs zu entdecken, so ist es möglich, anhand der bei der Cluster-Analyse ermittelten Ähnlichkeitsverteilung aufzudecken, welche Quellattribute besonders stark bzw. schwach die Klassenzugehörigkeit bestimmen.

Dieser Punkt und die Frage nach den Laufzeiten von CLUST_DB standen im Mittelpunkt des Abschnitts 3.4. Dort zeigte sich zunächst, daß die für den Ansatz gemachte Annahme, mit verhältnismäßig wenigen unterschiedlichen nominalen Werten rechnen zu müssen, für die Laufzeit wesentlich ist: Ist sie erfüllt, so kann mit einer raschen Bearbeitung des Programms gerechnet werden. Verletzt die Anzahl der zu gruppierenden Objekte diese, so steigt die Laufzeit von CLUST_DB enorm an. Im Anschluß daran konnte in Abschnitt 3.4.2 anhand einer Versuchsreihe mit C4.5 und Datensätzen der University of California (Irvine) die Tauglichkeit zur Dimensionsreduktion demonstriert werden.

3.6.2 Bemerkungen

In diesem Abschnitt werden einige Sachverhalte festgehalten, die zum Teil in den vorangegangenen Abschnitten nur angedeutet wurden oder im Sinne einer abschließenden Betrachtung nochmals wiederholt werden sollen.

Eines der grundlegenden Probleme der Cluster-Analyse liegt in der oft schwierigen Interpretation der Ergebnisse. Der Benutzer muß sich in der Regel damit zufrieden geben, die ermittelten Gruppierungen zu erhalten. Auf eine lesbare Beschreibung oder Charakterisierung der gefundenen Gruppierungen wird er vergeblich warten. Im günstigsten Fall vermittelt die Objekthierarchie einen so tiefgreifenden Blick in die Struktur der analysierten Daten, daß er selbst die Schlüsse ziehen kann — leider ein wahrlich selten eintretender Optimalfall. In vielen Anwendungen ist diese charakterisierende und durch den Benutzer interpretierbare Beschreibung unabdingbar. Diesem Aspekt trägt in besonderem Maße das *conceptual clustering* [MS83] Rechnung. Dort geht die Güte der Beschreibung eines Clusters mit in die Suche nach optimalen Gruppierungen ein. Diese Richtung wurde in der Arbeit nicht weiter verfolgt, sondern vielmehr die Auskunft darüber, welche Quellattribute den größten (bzw. kleinsten) Einfluß auf die Gestalt der Gruppen ausübten, als im Rahmen der Aufgabenstellung hinreichende Charakterisierung angesehen.

Wie bei jeder Cluster-Analyse steht und fällt auch das Ergebnis von CLUST_DB mit dem verwendeten (Ähnlichkeits-)Maß, das die Gruppierungsschritte steuert. Zur Arbeitshypothese wurde die Annahme gemacht, daß die zusammen in einer Tabelle vorkommenden Attribute (oder zumindest die daraus gewählten Quellattribute) semantisch verknüpft sind. Ist dies nicht Fall, so wird CLUST_DB schwerlich „sinnvolle“ Gruppierungen auffinden. Auf der anderen Seite wird man dies dann auch wohl von keinem anderen Verfahren erwarten dürfen.

Vorsicht ist jedenfalls dann angebracht, wenn eine Zirkularität beim Lernen dergestalt droht, daß ein Quellattribut in dem Zielattribut für Gruppierungen sorgt, die im Umkehrschluß zu einer scheinbar sauberen Unterteilung des Attributs B führen. Generell sollte man sich im klaren darüber sein, daß insbesondere die dominierenden Quellattribute in das Zielattribut „hineingerechnet" werden. So stelle man sich etwa vor, daß ein vorhandenes Klassifikationsattribut als Quellattribut B dafür genutzt wird, in einem anderen beliebigen Attribut A nach Gruppen zu suchen. Führt man diese Gruppen anschließend dem Lernen als neues Attribut A_{neu} hinzu, so erhält man mit sehr hoher Wahrscheinlichkeit ein ungemein aussagekräftiges Attribut, daß dem gelernten Ergebnis eine hochprozentige Korrektheit attestiert — für den Fall, daß $A \to B$ gilt, ist A_{neu} ein Attribut, das ganz allein mit einer Genauigkeit von 100% die Klasse determiniert!

Auch wenn numerische Attribute als Quellattribute gewählt werden können, so sollte man sich dessen bewußt sein, daß CLUST_DB diese symbolisch behandelt. Ist dies nicht erwünscht, so muß ein Diskretisierungsverfahren wie etwa NUM_INT (siehe Abschnitt 4.3) vorgeschaltet werden, das für eine Abbildung der einzelnen numerischen Werte auf symbolisch zu verarbeitende Intervalle sorgt.

Der eben bzgl. numerischer Attribute genannte Punkt und die bereits in Abschnitt 3.2.2 gemachte Feststellung, daß die gewählte Repräsentation nicht sonderlich vor Datenfehlern geschützt ist, leitet über zum nächsten Abschnitt, in dem mögliche Verbesserungen an CLUST_DB skizziert werden.

3.6.3 Verbesserungen

Es werden in diesem Abschnitt Verbesserungen diskutiert, die zwar angedacht, aber keineswegs gründlich ausgelotet, geschweige denn umgesetzt wurden. Dabei werden insbesondere die beiden letzten im Abschnitt 3.6.2 gemachten Bemerkungen aufgegriffen.

Behandlung numerischer Quellattribute

Um die Behandlung numerischer Attribute zu verbessern, muß das Maß dahingehend flexibilisiert werden, daß die Berechnung der Ähnlichkeitswerte für diese Attribute von den nominalen abgekoppelt wird. Dies kann dadurch geschehen, daß für numerische Quellattribute nicht mehr alle Werte in die Mitspielermengen aufgenommen werden, sondern vielmehr nur das gelesene Minimum und Maximum — jeweils für jeden Wert des Zielattributs. Die Überlappung der einzelnen Intervalle kann dann im folgenden dazu genutzt werden, die Ähnlichkeit zu bestimmen. Die nötigen Anpassungen in der Repräsentation für das Zusammenführen zweier Objekte sind naheliegend, es müssen lediglich die Grenzen des neuen Objektes so gewählt werden, das sie alle Werte der beiden zusammengeführten Objekte umschließen.

Neben dem Vorteil, daß dies den numerischen Charakter wahrt, hat dieses Vorgehen auch Vorzüge in Bezug auf die Laufzeit: In der Regel wird es so sein, daß der für die Laufzeit von CLUST_DB mitbestimmender Faktor $m_{B_{max}}$

— die maximale Anzahl vorkommender Werte der Quellattribute — durch ein numerisches Attribut gegeben ist, das sich oft um Größenordnungen von den nominalen Attributen unterscheidet. Obiger Ansatz vereinfacht die Ähnlichkeitsberechnung stark und dürfte in diesem Sinne die Laufzeit deutlich senken. Dabei wäre dann erneut abzuwägen, ob es sich im Sinne einer Reduktion der zu übermittelnden Tupel lohnt, aufwendigere Anfragen zu stellen, die bei einem numerischen Quellattribut für jeden Wert des Zielattributs direkt lediglich Minimum und Maximum anfordern.

Robustheit gegen Datenfehler

Im Abschnitt 3.2.2 wurde bereits ausgeführt, daß einem nur einmal vorkommenden (möglicherweise falschen) Wert aufgrund des Mengencharakters genau soviel Bedeutung beigemessen wird, wie den vielfach eingelesenen (korrekten) Werten. Dies ist in Anbetracht zu erwartender Datenfehler innerhalb realer Datenbanken nicht optimal. So zeigte es sich beispielsweise im Abschnitt 3.4.2, daß die Effektivität von CLUST_DB als Hilfsmittel zur Dimensionsreduktion deutlich mit der Zunahme von „Fehlern“ abnahm.

Der naive Ansatz, nicht nur die unterschiedlichen, sondern alle Werte in Mitspielerlisten (Mengen sind dies jedenfalls nicht mehr) aufzunehmen, muß an der Datenmenge scheitern: Der resultierende Platzbedarf entspricht dem der in Abschnitt 3.2.2 verworfenen Tupelrepräsentation.

Denkbar aber wäre es, zu jedem in den Mitspielermengen vorkommenden Werten zu speichern, wie oft dieser mit dem entsprechenden Wert im Zielattribut in der Datenbank auftritt. Der Platzbedarf wird davon kaum berührt, da maximal $n \cdot m_{B_{max}}$ ganze Zahlen zusätzlich verwaltet werden müssen.[29]

Auch die Anfragen an das DBMS sind leicht anzupassen — jedoch mit der unangenehmen Konsequenz, daß alle[30] Werte der Datenbank über das Netz bewegt werden müssen. Damit wäre der Vorzug CLUST_DBs, sehr wenig Werte über das Netz vom DBMS anzufordern, zunichte gemacht, was wesentlich schwerer wiegen dürfte als die Tatsache, daß das Ähnlichkeitsmaß entsprechender Anpassungen an die neue Repräsentation bedarf.

[29] n sei die Anzahl der Quellattribute und $m_{B_{max}}$ die maximale Anzahl vorkommender Werte in den Quellattributen.

[30] Bei Umsetzung der oben skizzierten Verbesserung bzgl. numerischer Quellattribute trifft dieser Punkt nur für nominale Attributwerte zu.

Kapitel 4

Aggregation in numerischen Attributen

Recht analog zum vorangegangenen Kapitel wird es auch hier darum gehen, Wertebereiche in Attributen zu finden, um sie einem Lernverfahren als Typ an die Hand zu geben. Während im dritten Kapitel die nominalen Attribute im Vordergrund standen, werden in diesem Teil der Arbeit die numerischen Attribute behandelt. Die konkrete Aufgabe lautet also, Intervalle zu bestimmen, die sich als sinnvolle Vergröberung der einzelnen Werte anbieten.

Davon abgesehen, daß es hier darum geht, ein Verfahren zu entwerfen, das auf einer relationalen Datenbank arbeiten kann, stellt dies eine Aufgabenstellung dar, die hinlänglich im Kontext des *maschinellen Lernens* bekannt ist: Ein Prozeß, bei dem (kontinuierliche) numerische Werte auf eine beschränkte Anzahl von Bereichen reduziert werden, denen dann symbolische Werte als Bezeichner zugeordnet werden können, wird als *Diskretisierung* bzw. *Kategorisierung* bezeichnet.

Ein solches Verfahren ist in doppelter Hinsicht wichtig: Zum einen kann ein Diskretisierungsverfahren zu einer enormen Datenreduktion führen, indem viele tausend Werte auf eine geringe Anzahl von Intervallen reduziert werden, die jeweils durch ein Symbol beschreibbar sind. Auf der anderen Seite eröffnet es vielen Lernverfahren überhaupt erst den Zugang zu einem Datensatz aus numerischen Werten, da vielfach in diesen verankert ist, daß der Wertebereich der erfaßten Attribute möglichst klein, aber in jedem Fall endlich zu sein hat.

Im Mittelpunkt dieses Kapitels steht der Abschnitt 4.2, in dem das zur Diplomarbeit entwickelte Verfahren NUM_INT mit seinen beiden integrierten Modi *Gap* und *Discontinuity* als Lösung der genannten Teilaufgabe vorgestellt wird. Diesem folgen die Besprechung der Einsatzmöglichkeiten und Experimente, die Auskunft über die tatsächliche Laufzeiten des Programmes geben. Das Kapitel endet schließlich mit einer Betrachtung anderer Verfahren, die auch praktisch in einer Versuchsreihe mit NUM_INT verglichen werden, und einer Diskussion dieses vierten Kapitels.

Doch zunächst sollen unterschiedliche grundsätzliche Herangehensweisen an die Aufgabe besprochen werden.

4.1 Ansatzmöglichkeiten

Es wird in diesem Abschnitt nicht primär darum gehen, einzelne ausgearbeitete Ansätze zu besprechen. Es werden vielmehr drei wesentliche Strategien auf dem Weg zum Entwurf des gewünschten Verfahrens skizziert und die jeweiligen Vor- und Nachteile herausgearbeitet.

Nachdem bereits bemerkt wurde, daß im dritten Kapitel eine ähnliche Zielsetzung verfolgt wurde, gilt es zunächst festzuhalten, daß CLUST_DB sich nicht als Diskretisierungsverfahren eignet — obwohl dieses sehr wohl numerische Attribute gruppieren kann. CLUST_DB tut dies jedoch, ohne Berücksichtigung des numerischen Charakters, rein symbolisch, und es sollte klar sein, daß als Resultat der Gruppierungsschritte auf keinen Fall Intervalle zu erwarten sind.

Generell bietet sich ein Cluster-Ansatz nicht an: Die Bildung von Intervallen müßte regelrecht dadurch erzwungen werden, daß in jedem Gruppierungsschritt nur diejenigen Objekte (Intervalle bzw. einzelne Werte) zusammengefaßt werden dürfen, die benachbart im Wertebereich zu finden sind. Ist dies etwa über einen entsprechenden Aufbau einer (degenerierten) Ähnlichkeitsmatrix möglich, in der lediglich benachbarte Objekte betrachtet werden, so bleibt das Problem, daß selbst eine derart reduzierte „Matrix“ $\mathcal{O}(N)$ Speicherplatz für sich beansprucht (mit N gleich der Anzahl der Werte).

Wendet man sich den eigentlichen Diskretisierungsverfahren zu, so erkennt man, daß sich insbesondere die Verfahren großer Beliebtheit erfreuen, die anhand einer Klassifikation Intervalle bilden. Man könnte also versuchen, zunächst eine im vorliegenden Fall an sich fehlende Klassifikation einzuführen, um anschließend ein solches Verfahren (als Beispiele seien etwa ChiMerge [Ker92], StatDisc [Ros95] und die Ansätze von [FI93] und [Paz95] genannt) auszuwählen, welches den Anforderungen in Bezug auf die zu erwartende Datenmenge entspricht und sich für die nötige Kopplung an eine relationale Datenbank anbietet. Dies ist freilich nicht trivial: Wie soll die Klassifikation herbeigeführt werden, welche Attribute könnten als Klassifikationsattribut dienen? Als wäre dies nicht schon unangenehm genug, gesellt sich dazu das Problem, daß die Intervallbildung dann nicht mehr „universell“ ist in dem Sinne, daß sie nicht mehr jeder Lernaufgabe gleichermaßen nützt. Für jeden veränderten Lernlauf dürfte eine neue Diskretisierung mit anderer Klassifikation fällig werden. Angesichts dieser Nachteile wurde diese Strategie nicht als gewinnbringend eingestuft.

Was schließlich bleibt, sind Diskretisierungsansätze, die allein aus dem vorhandenen Material der numerischen Werte Intervalle ableiten. Da keines dieser Verfahren bislang ohne Einschränkungen zu überzeugen wußte und überhaupt nur wenige als wirklich bekannt einzustufen sind, wird es in den folgenden Abschnitten darum gehen, einen neuen Ansatz ins Rennen zu schicken, diesen dem Vergleich mit den bekannten zu unterziehen und „Zwischenzeiten“ nach den ersten Runden zu nehmen.

4.2 NUM_INT

Nachdem im vorangegangenen Abschnitt verschiedene Ansätze mit ihren Vor- und Nachteilen diskutiert und eine Hauptstrategie favorisiert wurde, erfolgt in diesem Abschnitt die Vorstellung des erarbeiteten Verfahrens NUM_INT und der zugrundeliegenden Ansätze *Gap* und *Discontinuity*. Dazu werden beide Grundideen erläutert, deren algorithmische Umsetzung und die Implementation besprochen. Eine vergleichende Betrachtung anderer Verfahren erfolgt im Anschluß im Abschnitt 4.5.

Sowohl der *Gap*- als auch der *Discontinuity*-Ansatz arbeiten auf einer (zuvor) sortierten Wertefolge und durchsuchen diese aufsteigend nach *Trennstellen*. Lediglich die Auffassung, was eine Trennstelle darstellt, unterscheidet sie voneinander.

Gap: Bereits die Bezeichnung des Ansatzes — *gap* ist das englische Wort für *Lücke* — läßt erahnen, worauf er abzielt: Nachdem die Ordnung des numerischen Attributs hergestellt ist, wird in der Abfolge der Werte nach Lücken gesucht. Die größte Differenz zweier aufeinander folgender Werte teilt dabei das initiale Intervall $[min, max]$, die nächst kleineren Differenzen führen zu weiteren Unterteilungen in top-down Manier.

Discontinuity: Auch hier ist der Name Programm: Ist der Wertebereich klein im Vergleich zur Anzahl der Werte oder führen andere Umstände dazu, daß die Werte nahezu äquidistant in der Datenbank vorkommen, so macht es Sinn, die Häufigkeitsverteilung der Werte zu betrachten. Eine Diskontinuität in Bezug auf die Anzahl der Tupel pro Wert kann als Indiz für eine geeignete Trennstelle gewertet werden. Auch hier lassen sich die Kandidaten für eine Unterteilung ordnen, um ein zu Gap analoges Vorgehen zu ermöglichen. Als Ordnungskriterium wurde die absolute Differenz der Häufigkeiten zweier benachbarter Werte gewählt.

Die beiden verwandten Ansätze bringen insofern gute Voraussetzungen für die Bearbeitung realer Datenbanken mit, als daß bei festgelegter Anzahl der zu berücksichtigenden Trennstellen alle nötigen Informationen während eines Laufs durch die geordneten Attributwerte gesichert werden können. Dies wird in der folgenden Beschreibung des Algorithmus deutlich werden. Darüber hinaus wird der Benutzer mit einer Bereichshierarchie versorgt, die mehr Struktur trägt als eine bloße Partitionierung der Werte.

4.2.1 Algorithmus

Es werden im folgenden die beiden Algorithmen Gap und Discontinuity von NUM_INT beschrieben. Da sich die beiden bzgl. Initialisierung und Ausgabe nicht unterscheiden, wurde eine kompakte Darstellung gewählt, in der die gemeinsamen Teile nur einmal aufgeführt werden und die charakteristischen Stellen durch (B1) für Gap und (B2) für Discontinuity ausgewiesen werden.

Im zweiten Kapitel wurde ausgeführt, daß im relationalen Modell kein direkter Zugriff auf das i-te Tupel vorgesehen ist. Das hieße in diesem Fall, in dem nur das zu untersuchende numerische Attribut A von Interesse ist, daß nicht unmittelbar der i-te Wert angesprochen werden kann. Die in Abschnitt 3.2.5 beschriebene Schnittstelle zur Datenbank läßt es jedoch bequem zu, der Reihe nach Wert für Wert in Empfang zu nehmen. Da dies genau den nötigen Zugriffen entspricht, ist die im folgenden verwendete vereinfachende Sprechweise „lese den Wert a_i" in Bezug auf einen Zähler i gerechtfertigt. Es wird im Rahmen der Laufzeitabschätzung geklärt, wie sichergestellt wird, daß die Werte von A aufsteigend der Größe nach als $min(A) = a_1 \leq a_2 \leq \ldots \leq a_N = max(A)$ eingelesen werden.

Eingabe

- *Attribut A* — numerisches Attribut, das unterteilt werden soll.
- *Tiefe T* — Parameter, der die Tiefe der Hierarchie festlegt.

Ausgabe

- *binärer Baum* — Hierarchie der gefundenen Intervalle.

Datenstrukturen und ihr Platzbedarf

- $DIFF[1 \ldots i \ldots T] = (left, right, d)$
  Array, in dem die T größten Trennstellen zwischen benachbarten Werten $left, right$ bzgl. $d := a_i - a_{i-1}$ für Gap bzw. $d := |count(a_i) - count(a_{i-1})|$ für Discontinuity aufsteigend sortiert verwaltet werden.[1] $\rightarrow \mathcal{O}(T)$

- $MARKER[1 \ldots 2(T+1)] = (val, father)$
  Array, in dem Intervallgrenzen bzgl. val aufsteigend sortiert gespeichert werden. Über $father$ kann das zugehörige Intervall angesprochen werden. $\rightarrow \mathcal{O}(T)$

- $INTER[0 \ldots 2T] = (left, right, father)$
  Array, in das die gefundenen Intervalle $[left, right]$ der Reihe nach aufgenommen werden. Über $father$ kann das übergeordnete Intervall angesprochen werden. $\rightarrow \mathcal{O}(T)$

Algorithmus NUM_INT

(A) Initialisierung:

(1) $DIFF[1 \ldots T] := (\_, \_, \_)$
$MARKER[1] := (a_1, 0)$
$MARKER[2] := (a_N, 0)$
$INTER[0] := (a_1, a_N, N)$

(B1) Lokalisierung der T größten Lücken:

[1] $count(a)$ gibt an, wie oft der Wert a vorkommt.

(1) Zähler $i := 2$.
Lese den kleinsten Wert a_1.

(2) Lese den nächst größeren Wert a_i und berechne $d := a_i - a_{i-1}$.

(3) Sortiere den Eintrag (a_{i-1}, a_i, d) anhand von d in $DIFF$ ein. Sollte er wegen eines zu kleinen d keinen Platz unter den T größten Werten finden, ignoriere ihn.

(4) $i := i + 1$
Wenn $i \leq N$, dann gehe zu (2).

(B2) Lokalisierung der T deutlichsten Sprünge:

(1) Zähler $i := 2; c_1 := 0; c_2 := 0$.
$c_1 := count(a_1)$
$i := i + c_1$
Wenn $i > N$, dann gehe zu (C1).

(2) $c_2 := count(a_i)$
$d := |c_2 - c_1|$.

(3) Sortiere den Eintrag (a_{i-c_1}, a_i, d) anhand von d in $DIFF$ ein. Sollte er wegen eines zu kleinen d keinen Platz unter den T größten Werten finden, ignoriere ihn.

(4) $i := i + c_2$
$c_1 = c_2; c_2 = 0$.
Wenn $i \leq N$, dann gehe zu (2).

(C) Bestimmung der Intervalle:

(1) Zähler $i := 1$.

(2) Bestimme maximales n, so daß
$val(MARKER[n]) \leq left(DIFF[i]) \leq val(MARKER[n+1])$

(3) $INTER[2i-1] := (val(MARKER[n]),$
$left(DIFF[i]),$
$father(MARKER[n]))$
$INTER[2i] := (right(DIFF[i]),$
$val(MARKER[n+1]),$
$father(MARKER[n]))$
$father(MARKER[n]) := 2i - 1$
$father(MARKER[n+1]) := 2i$

(4) Sortiere $(left(DIFF[i]), 2i - 1)$ und $(right(DIFF[i]), 2i)$ in $MARKER$ ein.

(5) $i := i + 1$
Wenn $i \leq T$, dann gehe zu (2).

(C) Ausgabe der Intervalle:

(1) Ausgabe der Werte von $DIFF[1 \ldots T]$, $MARKER[1 \ldots 2(T+1)]$ und $INTER[0 \ldots 2T]$.

Korrektheit des Algorithmus

Die Korrektheit des Algorithmus wird durch vollständige Induktion über die Anzahl der abgearbeiteten Werte i von $DIFF[0 \ldots i \ldots T]$ gezeigt, indem bewiesen wird, daß für $0 \leq i \leq T$ gilt, daß die Eintragungen in $INTER[0 \ldots 2i]$ und $MARKER[1 \ldots 2(i+1)]$ korrekt sind.

Beweis Beweis durch vollständige Induktion über i:

Induktionsanfang für $i = 0$: Zu Beginn — es wurde noch keine Trennstelle bearbeitet — liegen alle Werte innerhalb des initialen Intervalls $[min, max]$ mit den einzigen Intervallgrenzen min und max und es gibt kein übergeordnetes „Vaterintervall". Dies entspricht gemäß (A1) genau den Werten für $INTER[0]$ und $MARKER[0], MARKER[1]$.

Induktionsschritt von $i \to i+1$: Seien also sowohl $INTER[0 \ldots 2i]$ und $MARKER[1 \ldots 2(i+1)]$ korrekt und $(left, right, diff) := DIFF[i+1]$ (mit $i+1 \leq T$) die zu bearbeitende Trennstelle.

(C2) sorgt dafür, daß in eindeutiger Weise das Intervall bestimmt wird, das aufzuspalten ist. Seien die Grenzen dieses Intervalls etwa durch $v_1 := val(MARKER[n])$ und $v_2 := val(MARKER[n+1])$ gegeben.

Um korrekt zu sein, müssen nun im wesentlichen zwei Dinge vom Algorithmus geleistet werden:

(1) Es müssen zwei neue Intervalle $[v_1, left], [right, v_2]$ erzeugt werden, wobei die Einordnung in die Hierarchie sichergestellt werden muß. Im vorliegenden Fall muß das zu $MARKER[n]$, $MARKER[n+1]$ gehörende Intervall als „Vater" der neuen Intervalle vermerkt werden. Dies leistet Schritt (C3), indem es die Intervalle $INTER[2(i+1)-1]$ und $INTER[2(i+1)]$ mit $INTER[father(MARKER[n])]$ als übergeordnetes Intervall erzeugt. Damit ist $INTER[0 \ldots 2(i+1)]$ korrekt.

Dafür, daß dies auch dann richtig ausgeführt wird, wenn $left$ auf eine Intervallgrenze fällt, sorgt (C2): Dort stellt die Suche nach dem maximalen n sicher, daß $left$ nur mit der linken Grenze ($MARKER[n]$) zusammenfallen kann. Man kann sich leicht davon überzeugen, daß dies dann keiner Sonderbehandlung bedarf.

(2) Zunächst muß gespeichert werden, daß $MARKER[n]$ und $MARKER[n+1]$ nun zum Intervall $2(i+1)-1$ bzw. $2(i+1)$ gehören. Dafür trägt (C3) Sorge. Des weiteren müssen die beiden neuen Grenzmarkierungen $left$, $right$ vermerkt werden. Dafür sorgt (C4), indem die beiden Werte und deren Referenz auf die gerade erzeugten Intervalle in $MARKER$ einsortiert werden — $left$ gehört zum Intervall $INTER[2(i+1)-1]$, $right$ zu $INTER[2(i+1)]$. Nach der Einsortierung ist demnach $MARKER[1 \ldots 2((i+1)+1)]$ korrekt.

Da beide Schritte korrekt ausgeführt werden, folgt aus der vollständigen Induktion die Korrektheit des Algorithmus. Daß er in jedem Fall in endlicher Zeit ausgeführt wird, ist offensichtlich: Nach der Initialisierung (A) und dem einmaligen Durchlauf aller numerischen Werte in (B) wird in (C) je einer der T Trennstellen in endlicher Zeit abgearbeitet. In (D) werden schließlich der Reihe nach die T Trennstellen und die daraus resultierenden endlich vielen Intervalle ausgegeben.

□

Laufzeitabschätzung für den Algorithmus NUM_INT

In der folgenden Auflistung der Kosten der einzelnen Schritte von NUM_INT wird das sortierte Lesen der einzelnen Attributwerte (B1, B2) zunächst ausgespart. Darüber hinaus werden die Schritte, die in konstanter Zeit ($\mathcal{O}(1)$) ausgeführt werden können, nicht gesondert aufgeführt.

(A) Initialisierung:

- (1) $\mathcal{O}(T)$ — trivial.

(B1) Lokalisierung der T größten Lücken:

- (1,4) $\mathcal{O}(T)$ Durchläufe von (2,3)
- (3) $\mathcal{O}(\log T)$ — wird in $DIFF$ ein *heap* verwaltet, so kann mit diesem Aufwand ein Wert unter T anderen einsortiert werden.[2]

(B2) Lokalisierung der T deutlichsten Sprünge:

- (1–4) $\mathcal{O}(N \log T)$ — insgesamt wird jeder Wert genau einmal betrachtet und dabei höchstens einmal in $DIFF$ einsortiert (s.o.).

(C) Bestimmung der Intervalle:

- (1,5) $\mathcal{O}(T)$ Durchläufe von (2) – (4)
- (2) $\mathcal{O}(\log T)$ — durch binäre Suche auf dem sortierten $MARKER$-Array.
- (4) $\mathcal{O}(T^2)$ — es sind T Werte in ein Array der Größe T einzusortieren.[3]

(C) Ausgabe der Intervalle:

- (1) $\mathcal{O}(T)$ — trivial.

Die Sortierung der Werte und das entsprechende Einlesen kann mit Hilfe der SQL-Anweisung `select A from R order by A` erfolgen. Dies führt zu einem Aufwand von $\mathcal{O}(N \log N)$, da N Werte zu sortieren sind, was etwa mit dem Einsatz von HEAPSORT [AHU83] erreicht werden kann. Die in Abschnitt 3.2.5 beschriebene Schnittstelle sorgt dafür, daß die sortierten Werte vom DBMS in konstanter Zeit pro Wert in Empfang genommen werden.

---

[2] Siehe etwa [AHU83].

[3] Der Einsatz eines heaps wäre möglich, würde aber (2) aufwendiger werden lassen und wurde angesichts des Gesamtaufwandes und der zu erwartenden Werte für T verworfen.

Lemma 4.1 *Sei N die Anzahl der numerischen Attributwerte, so ist die Laufzeit des Algorithmus* NUM_INT *sowohl für Gap als auch für Discontinuity für $T = \mathcal{O}(\log N)$ durch $\mathcal{O}(N \log N)$ beschränkt. Es genügt ein Datenbankzugriff, der einen Transfer von N Werten für Discontinuity und N' Werten für Gap erfordert, wobei N' die Anzahl der vorkommenden verschiedenen Werte von A bezeichnet.*

Beweis Die Abschätzung folgt aus den einzeln in der vorangegangenen Laufzeitabschätzung aufgeführten Positionen und dem bereits beschriebenen Zugriff auf die Daten:

$$\begin{aligned} &\mathcal{O}(\underbrace{T}_{Initialisierung} + \underbrace{N \cdot \log N}_{Datenbankzugriff} + \underbrace{N \cdot \log T}_{Gruppierung} + \underbrace{T \cdot (\log T + T)}_{Ausgabe}) \\ &= \mathcal{O}(N \log N + T^2) = \mathcal{O}(N \log N) \text{ für } T = \mathcal{O}(\log N) \end{aligned}$$

□

4.2.2 Implementation

In diesem Abschnitt werden kurz Details zur Implementation erörtert. Dies umfaßt die Erläuterung der Architektur und der Ausgaben an den Benutzer.

Der Rahmen, in dem NUM_INT implementiert wurde, deckt sich mit dem in Abschnitt 3.2.5 für CLUST_DB beschriebenen: Die Umsetzung erfolgte in C unter Benutzung der bereits erläuterten Schnittstelle zur Datenbank. Alle Attribute numerischen Typs können diskretisiert werden.

Die beiden Ansätze Gap und Discontinuity wurden dergestalt in NUM_INT vereint, als daß der gewünschte Modus durch Parameterübergabe an das Programm gewählt wird. Des weiteren kann bei Verwendung von Gap gewählt werden, ob alle Werte (*all*) oder nur die unterschiedlichen (*distinct*) betrachtet werden sollen. Offensichtlich resultiert beides in den gleichen Intervallen. Die Wahl von *distinct* reduziert jedoch unter Umständen die Laufzeit beträchtlich, da bei *all* vielfach identische Werte zeitaufwendig durch das Netz geschickt werden, um letztendlich intern sofort fallengelassen zu werden. Die Parameterwahl *all* ist insofern nur dann sinnvoll, wenn Wert darauf gelegt wird, daß die vom Programm gelieferte Information, an welcher Position getrennt wurde, sich auf die Anzahl aller Werte und nicht auf die Anzahl unterschiedlicher Werte beziehen soll.

Die Ausgabe des Programms — von Laufzeitinformationen abgesehen — gliedert sich in drei Abschnitte:

wichtigste Trennstellen: Es werden die wichtigsten T[4] Trennstellen inklusive der Auskunft, an der wievielten Stelle der Werteabfolge diese auftreten, ausgegeben. Bei Gap sind das die T größten Lücken; bei Verwendung von Discontinuity die T deutlichsten Übergänge bzgl. der Häufigkeiten, mit der benachbarte Werte gelesen wurden.

[4] T ist vom Benutzer anzugeben.

Intervallhierarchie: Die Intervallunterteilungen, die durch die oben beschriebenen T größten Trennstellen erzwungen wurden, werden der Reihe nach ausgegeben. Dabei wird auch angegeben, wieviele Werte sich in den einzelnen Intervallen befinden. Dabei zählt *all* alle, *distinct* nur die unterschiedlichen.

atomare Intervalle: Im letzten Teil werden die Intervalle ausgegeben, die während des Programmablaufs nicht weiter unterteilt wurden. Sie stellen eine Partition des Wertebereiches dar. Auch für diese wird die Anzahl von Werten innerhalb eines Intervalls ermittelt.

4.3 Einsatz von NUM_INT

NUM_INT ist ein Diskretisierungsverfahren, das konzipiert worden ist, um in numerischen Attributen relationaler Datenbanken eine Hierarchie von Intervallen ohne Rückgriff auf eine Klassifikation zu bilden. Der Benutzer kann sich dabei entweder der vollständigen Hierarchie oder der atomaren Intervalle bedienen, die das Programm ausgibt. Im ersten Fall kann der Benutzer startend mit dem gesamten Intervall bei Bedarf nach und nach die Intervalle in der von NUM_INT ermittelten Reihenfolge verfeinern. Besonders reizvoll ist diesbzgl. das Zusammenspiel von NUM_INT mit einem Verfahren, das in der Lage ist, die volle Information der Hierarchie zu nutzen, um beim Lernen eine geeignete Ebene zu finden. Dazu zählen etwa INDUCE [Mic83] und OTIS [Ker88]. Im anderen Fall können die atomaren Intervalle direkt als Partitionierung des Wertebereichs für eine Typdeklaration genutzt werden. In beiden Fällen ist die Anzahl der Intervalle durch den Parameter T festgelegt: Es lassen sich in der Hierarchie insgesamt $2T + 1$ Bereiche finden, von denen $T + 1$ *atomar* in dem Sinne sind, daß sie als nicht weiter unterteilte Intervalle die Blätter der Hierarchie darstellen.

Ist der Wertebereich groß im Vergleich zur Anzahl der Einträge, so sollte der Gap-Ansatz bevorzugt werden. Ist er verhältnismäßig klein, so daß nicht sinnvoll nach Lücken gesucht werden kann, so sollte Discontinuity bevorzugt werden.

Daß NUM_INT auf relationalen Datenbanken arbeiten kann, ist als besondere Eigenschaft und keinesfalls als Einschränkung zu verstehen: Es muß lediglich die Möglichkeit gegeben sein, sortiert der Reihe nach auf die Werte zugreifen zu können, oder diese im Speicher selbst zu sortieren.

Alle numerischen Werte einer realen Datenbank auf ihre Validität hin zu überprüfen, dürfte an der Datenmenge scheitern. Eine im Sinne obiger Ausführungen komprimierte Darstellung bietet hingegen Ansätze, die Qualität der Daten zu überprüfen, so daß NUM_INT auch in diesem Kontext genutzt werden kann.

4.4 Laufzeiten

In diesem Abschnitt wird es darum gehen, das Laufzeitverhalten von NUM_INT in der Praxis zu überprüfen. In erster Linie ist dabei die Abhängigkeit von der Anzahl der Werte von Interesse. Diese wird Aufschluß darüber geben, ob das Verfahren wie gefordert in der Lage ist, die großen Datenmengen realer Datenbanken zu bewältigen. Weiterhin wird überprüft, inwiefern die durch den Benutzer gewählte Tiefe die Laufzeit beeinflußt. Mit einem weiteren wichtigen Aspekt, der Frage nach der Güte der gefundenen Intervalle, befaßt sich Abschnitt 4.5.2.

4.4.1 Tupelabhängigkeit

Als Ausgangspunkt für die Beantwortung der ersten Frage wurden Tabellen ausgewählt, die (nicht nur in Bezug auf das Datenvolumen) dem *realen* Einsatz entsprechen.[5] Auf die gefundenen Intervalle wird nicht näher eingegangen, da hier lediglich die Laufzeit von Interesse ist — davon abgesehen, daß dies wegen der Vertraulichkeit der Daten auch nicht gestattet gewesen wäre.

Die Laufzeiten t_{all} und $t_{distinct}$ von NUM_INT in den beiden Modi Gap (all) und Gap (distinct) werden in Tabelle 4.1 aufgeführt.[6] Auf die Darstellung der Ergebnisse von Discontinuity wurde verzichtet, da sie erwartungsgemäß denen von Gap (all) entsprachen.

N	N'	t_{all}	$t_{distinct}$
62	13	1' 30"	1' 30"
840	21	1' 40"	1' 30"
2363	5	1' 40"	1' 30"
43777	79	3'	1' 30"
116862	116672	6'	6'
610114	120000	24'	7'
745781	125092	31'	9'

Tabelle 4.1: Laufzeiten für NUM_INT bei gleichbleibender Tiefe. N gibt die Anzahl aller Werte an, N' die der unterschiedlichen.

Tabelle 4.1 offenbart zweierlei: Die zuvor gestellte Frage, ob NUM_INT auf realen Datenbanken mit mehreren 100.000 Einträgen arbeiten kann, muß bejaht werden. Die wiedergegebenen Laufzeiten sind unter Berücksichtigung der bearbeiteten Datenmengen mehr als akzeptabel.

Des weiteren ist offenkundig, daß bei Beschränkung auf die unterschiedlichen Werte des zu untersuchenden Attributs durch den Modus *Gap (distinct)* die Laufzeit deutlich abnahm — und dies, obwohl NUM_INT intern für die Abarbeitung aufeinanderfolgender gleicher Werte praktisch keine Zeit benötigt, da

[5] Die Versuche wurden am Forschungszentrum Ulm der Daimler-Benz AG von P. Brockhausen (Vertrag Nr.: 094 965 129 7/0191) ausgeführt.

[6] Die Zeiten wurden auf einer SUN Sparc Station 10 mit Sparc 20 als DB-server ermittelt.

eine Differenz von 0 gar nicht erst weiter behandelt wird, sondern sofort der nächste Wert angefordert wird. Dies läßt den Schluß zu, daß der Transfer der einzelnen Werte über das Netz so zeitintensiv ist, daß dies die gesamte Laufzeit dominiert. Abbildung 4.1 verdeutlicht diesen Sachverhalt, in dem die benötigten Zeiten gegen die Anzahl der tatsächlich übermittelten Werte aufgetragen werden: Die lineare Abhängigkeit ist offensichtlich. Dies unterstreicht den Vorzug von NUM_INT, mit nur einem Lauf durch die Daten auszukommen.

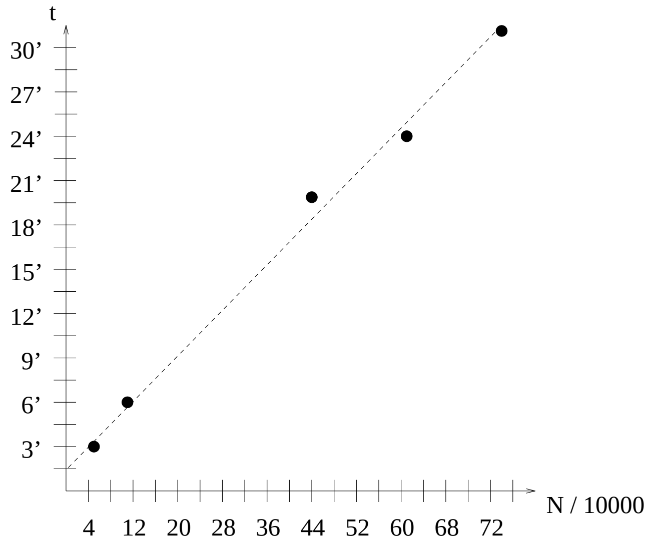

Abbildung 4.1: Laufzeit in Abhängigkeit der übermittelten Werte N

Liegt die Anzahl der unterschiedlichen Werte deutlich unter der Gesamtzahl, so wird Gap (distinct) deutlich schneller als die beiden anderen Möglichkeiten Gap (all) und Discontinuity sein. Spielt es keine Rolle, ob die von NUM_INT bestimmte Anzahl von Werten innerhalb eines Intervalls sich auf alle oder nur die unterschiedlichen bezieht, sollte bei Wahl von Gap der Parameter distinct benutzt werden.[7]

Die minimale Zeit von 1' 30" für die Bearbeitung von Attributen mit weniger als etwa 2.000 zu lesenden Werten entspricht der Zeit, die das Datenbankmanagementsystem vom Erhalt der SQL-Anfrage bis zum Ausliefern des ersten Wertes benötigte.

4.4.2 Tiefenabhängigkeit

Um zu klären, wie stark die Laufzeiten von der gewählten Tiefe T der Hierarchie abhängen, wurde ein numerisches Attribut einer Tabelle mit etwa 1000 Einträgen ausgewählt und für unterschiedliche T im Bereich von 10 − 1000 diskretisiert. Dabei konnte praktisch keine Zunahme in der Laufzeit beobachtet werden. Die Zeit in der Größenordnung von einigen Sekunden, die beim

[7] Es sei daran erinnert, daß die Intervallbildung davon nicht berührt wird.

Start von NUM_INT mehr benötigt wurde, geht auf die entsprechend gestiegene Anzahl von Systemaufrufen zur Bereitstellung des Speicherplatzes zurück. Dies kann sich selbstverständlich ändern, wenn T (unrealistisch) groß auf einen Wert gesetzt wird, der zu einem Speicherplatzbedarf für die Datenstrukturen führt, der den ausführenden Rechner überfordert.

Dieses in Bezug auf die effektive Laufzeit gutmütige Verhalten ist besonders erfreulich, da es dem Benutzer erlaubt, T deutlich größer zu wählen, als es die Anwendung zu erfordern scheint. So kann bei Bedarf auf die „Reserve“ zurückgegriffen werden, ohne einen neuen Lauf starten zu müssen.

4.5 Vergleich mit anderen Verfahren

Im diesem Abschnitt werden andere Verfahren vorgestellt, die dabei mit NUM_INT verglichen werden. Dies beschränkt sich nicht nur auf die Betrachtung der zugrunde liegenden Ansätze; es sollen vielmehr auch experimentelle Vergleiche angestellt werden.

4.5.1 Verwandte Ansätze

Diskretisierungsverfahren lassen sich leicht in zwei Kategorien einteilen: Die eine beinhaltet diejenigen Ansätze, die bei der Bildung der Intervalle eine vorhandene Klassifikation berücksichtigen. Diese werden häufig im Englischen als *class driven* bezeichnet, was in diesem Fall getrost wörtlich als „klassengesteuert“ übersetzt werden darf.

In der anderen befinden sich dementsprechend Verfahren, die ohne eine Klassifikation ihren Dienst verrichten. Sei es, daß eine solche nicht vorliegt oder (unvernünftigerweise) bei der Diskretisierung vernachlässigt wird.

In den letzten Jahren ist eine Vielzahl von klassengesteuerten Verfahren vorgestellt worden und ihre Überlegenheit im Vergleich zu den letztgenannten Ansätzen in Anwesenheit eines Klassifikationsattributs demonstriert worden, die in diesem Zusammenhang dann oft „klassenblind“ genannt werden.[8]

In der Tat stellt die Kenntnis einer Klassenzugehörigkeit einen immensen Wissensvorteil dar, bietet sie doch Ansatzpunkte für informationstheoretische Überlegungen. Es sollte jedoch bedacht werden, daß der vermeintliche Defekt, ohne Rückgriff auf eine Klassifikation zu diskretisieren, freilich dann zum Vorteil gereicht, wenn eine solche nicht vorhanden ist. Klassengesteuerte Verfahren sind dann schlichtweg nicht einsetzbar, da sie nicht nur die Klassenzugehörigkeit der Objekte *berücksichtigen*, sondern vielmehr auf sie *angewiesen* sind.

Im Rahmen der Aufgabenstellung liegt eine Ausgangssituation vor, die keinen Rückgriff auf ein Klassifikationsattribut erlaubt. So sollen in diesem Abschnitt nur die beiden aus der Statistik bekannten EQUAL-WIDTH und EQUAL-FREQUENCY und der jüngst in einer Diplomarbeit am LS8 ([Wes95]) skizzierte Ansatz von S. Wessel (im folgenden *Delta-Suche* genannt[9]) mit NUM_INT verglichen werden. Die beiden erstgenannten wurden ausgewählt, da sie weithin

[8] Siehe etwa [Ros95].

[9] In der erwähnten Arbeit ist dem kurz beschriebenen Ansatz kein Name vergeben worden.

bekannt sind — sie werden in der Literatur stets zu einem Vergleich herangezogen, wenn es darum geht, ein neues Diskretisierungsverfahren vorzustellen — und trotz vieler guter klassengesteuerter Verfahren noch eingesetzt werden. Der Ansatz von S. Wessel wird betrachtet, da er durchaus als Ausgangspunkt für die Entwicklung von NUM_INT zu bezeichnen ist.

Equal-Width

Der EQUAL-WIDTH-Ansatz sieht vor, die vorkommenden N Werte im Bereich $[x_{min}, x_{max}]$ so in eine vom Benutzer anzugebende Zahl n von Intervallen zu teilen, daß jedes dieser erzeugten Teilintervalle $[T_i, T_{i+1}]$ die gleiche Größe hat. Die einzelnen Trennpunkte T_i mit $i = 0 \dots n$ liegen demzufolge bei

$$T_i := x_{min} + i \cdot (x_{max} - x_{min})/n$$

Das Verfahren wird dadurch geprägt, daß außer dem Minimum x_{min} und dem Maximum x_{max} des zu diskretisierenden Attributs keine weiteren Informationen nötig sind bzw. ausgewertet werden. Dies ist in Bezug auf Einfachheit und die zu erwartende Laufzeit bestechend: Es ist keine Sortierung erforderlich, da Minimum und Maximum durch einen Lauf durch die Daten bestimmt werden können, was einer Komplexität von $\mathcal{O}(N)$ entspricht. Sind diese beiden Werte bereits bekannt oder liegt bereits eine Sortierung vor, dann muß auf keinen bzw. zwei Attributwerte zugegriffen werden.

Die Kehrseite der Medaille ist ebenso schnell ausgemacht — sie folgt unmittelbar aus der oben beschriebenen Eigenschaft: Werden ausschließlich Minimum und Maximum der Folge betrachtet, geht das Verfahren im Gegensatz zu NUM_INT in keiner Weise auf die Verteilung der einzelnen Werte ein. Es wird vielmehr implizit die Gleichverteilung der Werte über den gesamten Bereich angenommen. Daß dieses Potential nicht ausgeschöpft wird, kann sich u. a. darin rächen, daß Intervalle gebildet werden, in denen kein einziger Wert liegt. Dies kann in dieser Form bei Verwendung von NUM_INT nicht passieren: Reichen die gefundenen Trennstellen nicht aus, um die vom Benutzer gewünschte Anzahl an Intervallen zu erzeugen, so führt dies nicht dazu, daß (sinn)leere Intervalle hinzugefügt werden. Es werden vielmehr alle ermittelten Trennstellen genutzt und der Benutzer darüber informiert, daß seine Wahl nicht angemessen war.

EQUAL-WIDTH ist darüber hinaus nicht wie NUM_INT in der Lage, eine Hierarchie von Intervallen zu erzeugen. Ganz besonders deutlich wird der Nachteil, wenn ein Lernverfahren diese Hierarchie selbständig dahingehend ausnutzt, daß es beim Lernprozeß steuert, auf welche Intervalle im Hinblick auf ein gutes Ergebnis zurückgegriffen wird. Bietet das verwendete Lernverfahren dies nicht an, wird diese Arbeit auf den Benutzer verlagert, aber der eigentliche Vorteil bleibt: Erkennt man bei Verwendung von EQUAL-WIDTH, daß das Resultat zu wünschen übrig läßt, so bleibt nichts anderes übrig, als eine neue Diskretisierung mit einem anderen Parameter n für die Anzahl der gewünschten Intervalle anzufordern. Wird NUM_INT benutzt, so kann man sich einer anderen Ebene der bereits bestimmten Hierarchie bedienen.

Equal-Frequency

Während die zuvor beschriebene Methode die Intervalle in gleich große Bereiche zerlegt, führt EQUAL-FREQUENCY die Diskretisierung so aus, daß in jedem Abschnitt (möglichst) gleich viele Werte vorkommen.

Anders als bei EQUAL-WIDTH findet also die Werteverteilung bei diesem Verfahren Berücksichtigung. Damit ist EQUAL-FREQUENCY aber auch auf eine Sortierung der zu unterteilenden Werte angewiesen und so mit einer Laufzeit von $\mathcal{O}(N \log N)$[10] aufwendiger als das zuvor diskutierte EQUAL-WIDTH und unterscheidet sich praktisch nicht von der für NUM_INT im Abschnitt 4.2.1 hergeleiteten Laufzeit, die (für eine entsprechende Tiefe der Hierarchie) ebenfalls vom Aufwand der Sortierung bestimmt wird.

Auch wenn dies angesichts des Bestrebens von EQUAL-FREQUENCY, gleichmäßig besetzte Intervalle zu bilden, überraschend erscheint, ist es doch wie bei Verwendung von EQUAL-WIDTH auch hier möglich, daß einige der erzeugten Bereiche leer bleiben: Kommen so viele gleiche Werte vor, daß die für ein Intervall vorgesehene Kapazität überschritten wird (ein Schnitt kann nicht erfolgen, da gleiche Werte nicht auf verschiedene Intervalle verteilt werden dürfen), kann es schnell passieren, daß am Ende nicht mehr genügend Werte da sind, um die verbleibenden Intervalle zu füllen. Anders als NUM_INT ist EQUAL-FREQUENCY wie auch EQUAL-WIDTH nicht in der Lage, auf diesen Fall zu reagieren. Es bleibt dem Benutzer überlassen, die gebildeten Bereiche zu inspizieren und bei Bedarf entweder die Anzahl der Intervalle kleiner zu wählen oder die Grenzen „von Hand“ zu verschieben.

Wie EQUAL-WIDTH bildet auch EQUAL-FREQUENCY keine Bereichshierarchie.

Delta-Suche

In [Wes95] wird ein Ansatz skizziert, der in ein Diskretisierungsverfahren umzusetzen ist: Es wird vorgeschlagen, die Werte aufsteigend der Größe nach zu sortieren, um in der Abfolge dieser die Differenzen der benachbarten Werte mit einem festzulegenden Schwellenwert δ zu vergleichen. Die Werte $x_1 \leq x_2 \leq \ldots x_i \leq x_{i+1}$ werden solange zu einem Intervall zusammengefaßt, wie die Differenz zweier aufeinanderfolgender Werte kleiner als δ ist. Eine größere Differenz $d = x_{i+1} - x_i$ zeigt eine Trennstelle an und bewirkt, daß das aktuelle Intervall mit x_i geschlossen und ein neues mit x_{i+1} eröffnet wird.

Der Reiz dieses einfachen Verfahrens liegt in dem offensichtlichen inkrementellen Charakter: Jeder Wert wird nur einmal begutachtet und danach „fallengelassen“, d. h. nicht intern gespeichert. Dies ist in Anbetracht des geringen Platzbedarfs — lediglich die Information über die bereits gebildeten Intervalle und die aktuell betrachteten x_i und x_{i+1}, sind zu speichern — und die lineare Laufzeit in Bezug auf die Anzahl der Werte N verheißungsvoll.

Der größte Nachteil dieses Ansatzes ist die problematische Wahl des für die Intervallbildung entscheidenden Parameters δ: Von der trivialen Beobachtung abgesehen, daß kleinere (größere) Werte für δ wahrscheinlich zu einer größeren

[10] N sei die Anzahl der vorhandenen numerischen Werte.

(kleineren) Zahl n von Bereichen führen, ist nichts Hilfreiches für die Wahl von δ offensichtlich. Da wiegt es besonders schwer, daß das Verfahren nicht in der Lage ist, eine Hierarchie von Intervallen aufzubauen: Während bei den bereits besprochenen Verfahren EQUAL-WIDTH und EQUAL-FREQUENCY im Falle eines Mißerfolgs schnell abschätzbar sein sollte, um welchen Betrag n zu verändern ist, dürfte die Anpassung von δ ungleich schwerer fallen.

4.5.2 Experimenteller Vergleich

Die Beantwortung der Frage, wie nützlich die von NUM_INT gefundenen Intervalle für Lernverfahren im allgemeinen sind, fällt ungleich schwerer als die bereits im Abschnitt 4.4 diskutierte Frage nach den Laufzeiten. Beschränkt man sich jedoch auf Verfahren, die aus klassifizierten Beispielen lernen, so wird die Güte einer Diskretisierung und damit auch des zugehörigen Verfahrens anschaulich: Je geringer die Fehlerquote des gelernten Resultates ist, desto hilfreicher waren die zuvor bestimmten Intervalle.

Die in diesem Abschnitt zu beschreibende Versuchsreihe folgt diesem Ansatz, indem C4.5 [Qui93] darauf angesetzt wurde, anhand unterschiedlicher Datensätze mit gegebener Klassifikation jeweils einen Entscheidungsbaum zu bestimmen. Dies geschah mit unterschiedlicher Diskretisierung der vorhandenen numerischen Attribute in direktem Vergleich zu den beiden Verfahren EQUAL-WIDTH (EW) und EQUAL-FREQUENCY (EF).

Es sei an dieser Stelle darauf hingewiesen daß C4.5 im allgemeinen (die Versuche werden dies bestätigen) ohne externe Diskretisierung zu sehr guten Ergebnissen kommt. Daß die Wahl dennoch auf C4.5 fiel, hängt mit dessen Verbreitung und Bekanntheit zusammen: C4.5 ist leicht zu handhaben und anerkanntermaßen erfolgreich, so daß in vielen aktuellen Publikationen über Diskretisierungsverfahren dieses als „Meßlatte“ angelegt wird.

Nachteilig an dieser Wahl ist, daß der zuvor herausgestellte besondere Vorteil von NUM_INT sich nicht auswirken konnte: C4.5 bietet nicht die Möglichkeit, eine gelernte Wertehierarchie auszunutzen. Dies muß bei der Diskussion der Versuchsergebnisse entsprechende Berücksichtigung finden.

Als Datensätze diente das bereits im Abschnitt 3.4.2 beschriebene Material der University of California (Irvine), so daß die vorgestellten Ergebnisse im Sinne obiger Ausführung den Vergleich mit Versuchsreihen andere Veröffentlichungen zusätzlich erleichtert.

In Tabelle 4.2 werden die von C4.5 in den unterschiedlichen Domänen ermittelten Fehlerraten der gelernten Entscheidungsbäume bei Verwendung eines der oben genannten Diskretisierungsverfahren wiedergegeben. Die Anzahl der Intervalle wurde für alle Datensätze und Verfahren gleichermaßen auf 5 festgesetzt.

Die Tabelle 4.2 liefert insofern kein einheitliches Bild, als daß sie kein Verfahren in allen Domänen vorne sieht. Bei den Datensätzen *Hepatitis* und *Labor-neg.* liegen die ermittelten Fehlerraten für die getesteten Verfahren in etwa der gleichen Größenordnung. Bei *Glass* und *Iris* schneidet NUM_INT (Gap) schlecht ab, hier scheint der Ansatz, nach Lücken in den numerischen Attributen zu suchen, deplaziert.

Domäne	—	Gap	Disc.	E.-Width	E.-Frequency
Iris	1.3%	10.7%	2.7%	4.7%	3.3%
Glass	5.6%	45.0%	13.6%	31.1%	9.3%
Hepatitis	5.2%	6.5%	6.5%	7.1%	3.9%
Labor-neg.	2.5%	5.0%	2.5%	0.0%	0.0%

Tabelle 4.2: Von C4.5 ermittelte Fehlerraten bei verschiedenen Diskretisierungen. In der zweiten Spalte (—) sind die ohne Diskretisierung erzielten Ergebnisse aufgeführt.

Es mag im ersten Augenblick enttäuschen, daß Num_Int nicht über die beiden anderen Verfahren dominiert. Bedenkt man aber die eingangs gemachte Bemerkung, daß der besondere Vorsprung Num_Ints gegenüber Equal-Width und Equal-Frequency in der Versuchsanordnung nicht zum tragen kommen konnte, darf das Ergebnis als positiv gewertet werden — zumal es sich bei den beiden genannten um die bekanntesten Verfahren handelt.

Ganz allgemein muß die Auswertung solcher Versuche berücksichtigen, daß die erzielten Ergebnisse aufgrund des Ansatzes „klassenblinder“ Verfahren, ausschließlich aus den numerischen Daten zu lernen, unter Umständen mehr über die verwendeten Datensätze als über die Verfahren aussagen, indem deutlich wird, ob die implizit gemachte Annahme über die Verteilung der Werte angemessen für die jeweilige Domäne war. Es lassen sich leicht Beispiele konstruieren, die die Überlegenheit eines der genannten Verfahren zu dokumentieren scheinen. Insofern stellen die beiden Modi von Num_Int eine Ergänzung zu den genannten Ansätzen dar.

Der Frage, welche Ergebnisse für eine andere Wahl bzgl. der Anzahl der zu bestimmenden Intervalle erzielt werden, wurde nicht nachgegangen. Aus den Beschreibungen der unterschiedlichen Ansätze geht jedoch hervor, daß sich Num_Int wesentlich robuster als die genannten anderen Verfahren verhalten sollte: Dadurch, daß die Trennstellen bei Num_Int der (vermeintlichen) Priorität nach geordnet sind, kann es anders als bei den Verfahren Equal-Width und Equal-Frequency nicht zu einer besonders ungünstigen Wahl bzgl. der Anzahl der zu bestimmenden Intervalle kommen. Die Erhöhung der zu erzeugenden Intervalle um 1 bewirkt etwa bei Equal-Width, daß alle Intervalle verändert werden. Bei Num_Int wird lediglich ein Intervall durch zwei neue Teilintervalle ersetzt.

Es sei an dieser Stelle abschließend daran erinnert, daß in diesen Versuchsreihen nur ein Teilgebiet des möglichen Einsatzes von Num_Int beleuchtet wurde, das besonders anschauliche Ergebnisse lieferte. In Bezug auf den primären Einsatz auf Daten ohne Klassifikation sei bemerkt, daß Versuche im Rahmen eines Zusammenspiels mit Rdt/db in Mobal [MWKE93] ebenfalls positiv verliefen: So führte die Einführung der von Num_Int bestimmten Intervalle dazu, daß auf den bereits beschriebenen Daten der Daimler-Benz AG Regeln gelernt wurden, deren Korrektheit sich zwischen 79 und 84 Prozent bewegte. Ohne Zuhilfenahme der Intervalle lag diese bei maximal 63%.

4.6 Diskussion

Wie bereits das vorangegangene Kapitel soll auch das vierte Kapitel, in dem es um die Behandlung numerischer Attribute ging, abschließend diskutiert werden. Diese Diskussion wird zum einen dazu genutzt, den vierten Teil der Arbeit zusammenzufassen; zum anderen sollen aber auch ergänzende Bemerkungen und Verbesserungsvorschläge gemacht werden.

4.6.1 Zusammenfassung

Wurden im dritten Kapitel nominale Attribute behandelt, so galt das Interesse des vierten Teils der Arbeit den numerischen Attributen. Dabei wurde eine Ausgangssituation angetroffen, die aus mehreren Gründen ein anderes Vorgehen als im dritten Kapitel erforderte. Einer dieser Gründe bestand darin, daß im Gegensatz zur Behandlung nominaler Attribute hier mit sehr vielen unterschiedlichen Werten zu rechnen ist.

Es lag vielmehr eine Aufgabenstellung vor, die auf den Entwurf eines neuen Diskretisierungsverfahren abzielte, das ohne Klassifikation aus den einzelnen Werten eines numerischen Attributs einer relationalen Datenbank Intervalle bilden sollte. Die grundlegende Idee, in der Abfolge aufsteigend sortierter Werte nach Trennstellen zu suchen, um ein top-down Vorgehen zu ermöglichen, das mit nur einem Lauf durch die Daten eine Hierarchie von Intervallen erzeugen kann, wurde durch den Algorithmus NUM_INT realisiert. In diesem sind zwei grundsätzliche Arbeitsmodi eingebettet worden: Gap interpretiert große Differenzen benachbarter Werte als geeignete Stelle für eine Unterteilung, Discontinuity sieht eine solche in einer deutlichen Schwankung der Häufigkeiten, mit denen aufeinander folgende Werte gelesen werden.

Der Vergleich im Abschnitt 4.5 ergab, daß NUM_INT zwar nicht in der Lage ist, die beiden bekanntesten Verfahren EQUAL-WIDTH und EQUAL-FREQUENCY in Bezug auf die Güte der bestimmten Intervalle abzulösen. Es sollte jedoch deutlich geworden sein, daß NUM_INT sehr wohl als Ergänzung anzusehen ist, da der Erfolg jedes der vorgestellten Diskretisierungsverfahren ohne Klassifikation sehr stark von der Verteilung der Werte abhängt und sich NUM_INT bzgl. des Ansatzes deutlich von den anderen Verfahren unterscheidet. Ein deutlicher Vorteil von NUM_INT liegt in der Fähigkeit, eine Hierarchie von Intervallen zu bestimmen. Dieser Vorteil kann in der Praxis z. B. durch den Einsatz eines Verfahrens ausgespielt werden, das die volle Information einer Hierarchie auszunutzen vermag kann wie etwa INDUCE [Mic83] und OTIS [Ker88]. Dieser Vorsprung kam in der Versuchsreihe des Abschnitts 4.5.2 nicht zum tragen, da das verwendete C4.5 trotz vieler positiver Eigenschaften einen solchen Vorzug nicht bietet.

Die für den primären Einsatz auf numerischen Attributen einer realen Datenbank entscheidende Frage nach den Laufzeiten von NUM_INT konnte im Abschnitt 4.4 sehr zufriedenstellend beantwortet werden. Es ergab sich dabei, daß die Laufzeit praktisch linear mit der Anzahl der übermittelten Werte ansteigt. Dies unterstreicht zum einen den Vorzug NUM_INTs mit nur einem Lauf durch die Daten auszukommen und motivierte den Einsatz eines Parameters, der bei

Verwendung von Gap das Datenbankmanagementsystem anweist, nur die für die Intervallbildung nötigen *unterschiedlichen* Werte zu liefern.

Ähnlich wie bei CLUST_DB sind auch bei NUM_INT die grundlegenden Ansätze nicht an eine relationale Datenbank gebunden. Es muß lediglich dafür gesorgt werden, daß auf die Werte in aufsteigend sortierter Reihenfolge zugegriffen werden kann.

4.6.2 Bemerkungen

Im Abschnitt 4.5 wurde deutlich, daß jedes der beschriebenen Verfahren unterschiedliche Annahmen über die Verteilung der Werte macht. Treffen diese für die zu untersuchenden Daten zu, so sind alle diese Verfahren in der Lage, für den Betrachter besonders suggestive Intervalle zu bilden.

Wie bei EQUAL-WIDTH und EQUAL-FREQUENCY kann es auch bei NUM_INT vorkommen, daß keine sinnvollen Intervalle gefunden werden können. Neben dem nicht betrachtenswerten Fall, daß alle Werte auf einen Punkt zusammenfallen, kann es in einem anderen Extremfall vorkommen, daß Discontinuity keine einzige Unterteilung vornimmt: Sind die Werte des numerischen Attributs homogen über den gesamten Wertebereich verteilt, so ist kein Sprung in der Häufigkeitsverteilung zu finden. Lücken sind sehr wohl gegeben, führen aber nur zur einzelnen Separation der ersten unterschiedlichen Werte. Bei einer solchen Verteilung der Werte scheint es aber auch keine vernünftige Alternative zu EQUAL-WIDTH bzw. EQUAL-FREQUENCY zu geben.

Daß sich Trennstellen nach deren Priorität ordnen lassen, ist wesentlich für den top-down Charakter von NUM_INT. Dies hat eine weitere positive Konsequenz, die bislang nur kurz in Abschnitt 4.5.2 erwähnt wurde: Dadurch, daß in jedem Fall die Trennstellen ihrer Bedeutung nach ausgewertet werden und diese zu je *einer* Aufspaltung eines Intervalls in zwei Teilintervalle führen, kann von einer gleichmäßigen „Konvergenz" des Verfahrens insofern gesprochen werden, als es nicht wie bei EQUAL-WIDTH und EQUAL-FREQUENCY vorkommen kann, daß bei Erhöhung der Anzahl zu bildender Intervalle um 1 fast alle bislang gültigen Grenzen verschoben werden. Bei den Versuchen bzgl. der Güte der unterschiedlichen Diskretisierungsverfahren mit $n = 5$ Intervallen, wäre bei der Wahl von $n = 6$ Intervallen für NUM_INT in jedem Fall ein mindestens genauso gutes Ergebnis erzielt worden. Dies kann keines der anderen beteiligten garantieren, wie etwa Abbildung 4.2 für EQUAL-WIDTH illustriert. In diesem Zusammenhang ist bei Versuchsreihen mit EQUAL-WIDTH und EQUAL-FREQUENCY zu beachten, daß wenn mehr als ein n_1 mit dem Ziel betrachtet werden soll, allgemeinere Ergebnisse zu erzielen, diese nicht als Vielfaches von n_1 gewählt werden sollten. Leider wird dieser Punkt oft nicht bedacht, so daß typischerweise zwei Versuchsreihen durchgeführt werden mit $n = 5$ und $n = 10$.

Auf der anderen Seite ist NUM_INT deutlich sensitiver gegenüber dem Einfügen neuer Werte in das numerische Attribut als es die anderen Verfahren sind: Fügen sich neue Werte so in den Wertebereich ein, daß die Trennstellen zunichte gemacht werden, so kann ein neuer Start von NUM_INT bei nur wenigen neuen Werten schon zu einer deutlich veränderten Hierarchie führen. Zudem sollte beim Einfügen neuer Werte bedacht werden, daß NUM_INT in dem Sinne

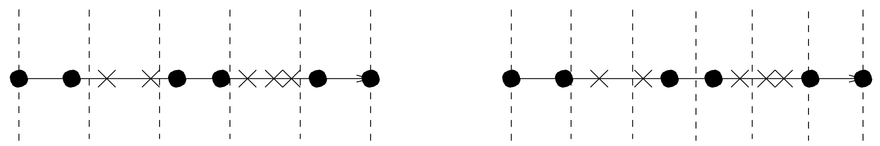

Abbildung 4.2: Links: In Bezug auf die unterschiedlichen Klassen ($\times, \bullet$) perfekte Diskretisierung in $n = 5$ Intervalle mit EQUAL-WIDTH. Rechts: Deutlich schlechtere Aufteilung für $n = 6$.

„scharf" trennt, daß die Grenzen genau auf vorhandene Werte gesetzt werden. Obwohl alle zum Zeitpunkt der Diskretisierung bekannten Werte umfaßt werden, ergibt die Vereinigung aller gebildeten atomaren Intervalle nicht den gesamten Wertebereich, sondern es werden vielmehr Bereiche ausgespart, in denen kein Wert lag. Fällt nun ein neuer Wert in genau einen dieser Zwischenbereiche, so wird er etwa von einer aus den Intervallen abgeleiteten Typdeklaration nicht erfaßt. Eine entsprechende Änderung des Trennverhaltens von NUM_INT ist jedoch ohne Seiteneffekte möglich.

4.6.3 Verbesserungen

Im folgenden werden kurz Verbesserungsvorschläge festgehalten, die zwar angedacht, aber bislang nicht umgesetzt wurden. So ist es möglich, NUM_INT anzuweisen, nicht den gesamten Wertebereich sondern lediglich einen Ausschnitt davon zu untersuchen. Dies macht insbesondere dann Sinn, wenn viele Trennstellen außerhalb eines vom Benutzer fokussierten Bereiches liegen und damit der Wert T für die Tiefe der Hierarchie unübersichtlich hoch gewählt werden müßte, um den eigentlich interessierenden Bereich zu unterteilen.

Sowohl bei Gap als auch bei Discontinuity werden einfache mathematische Modelle zugrunde gelegt. Beim erstgenannten Modus wird als Trennfunktion $f_g(x_i, x_{i+1}) := x_{i+1} - x_i$ beim anderen $f_d(x_i, x_{i+1}) := |c(x_{i+1}) - c(x_i)|$ verwendet, wobei $c(x)$ angibt, wie oft der Wert x vorkommt. Bei beiden sind viele andere Funktionen denkbar wie etwa bei Gap eine prozentuale statt absolute Differenz. Gerade in Bezug auf die beiden Modi Gap und Discontinuity wäre es ausgesprochen reizvoll, eine Funktion bzw. ein Maß zu finden, das noch mehr auf das Profil der Werteverteilung eingeht und mehr von einzelnen Werten abstrahiert. Auf diesem Wege wäre vielleicht auch eine Vereinheitlichung anstelle der vorliegenden Modulbauweise zu erzielen. Sicher ein interessantes, aber auch schwieriges Unterfangen, da der charakteristische quasi-inkrementelle Charakter NUM_INTs in jedem Fall angesichts der Laufzeiten gewahrt werden sollte.

Kapitel 5

Zusammenfassung

In der Arbeit wurden zwei Verfahren, deren zugrundeliegenden Ansätze und deren Implementation vorgestellt:

CLUST_DB bildet in nominalen Attributen einer relationalen Datenbank mit einer bottom-up Cluster-Analyse eine Wertehierarchie, die z. B. von Lernverfahren auszunutzen ist. Benutzt wird dabei ein in dieser Arbeit eingeführtes Ähnlichkeitsmaß, das es CLUST_DB als bislang einzigem Verfahren erlaubt, eine Cluster-Analyse direkt über die Werte eines ausgewählten Zielattributs einer Tabelle einer relationalen Datenbank durchzuführen. Die Wahl der für das Ähnlichkeitsmaß benötigten Quellattribute konnte dabei in Zusammenhang mit dem Konzept der funktionalen Abhängigkeiten gebracht werden.

Neben der Lösung des in der Aufgabenstellung geforderten Entwurfs eines Verfahrens zur Auffindung interessanter Wertebereiche wurde ferner in Abschnitt 3.3.6 der Einsatz von CLUST_DB zur Dimensionsreduktion für klassifizierte Daten erläutert und dessen Tauglichkeit im Abschnitt 3.4.2 experimentell unter Beweis gestellt.

Mit NUM_INT wurde ein Diskretisierungsverfahren für numerische Attribute relationaler Datenbanken erarbeitet und vorgestellt, das wie die aus der Statistik bekannten Ansätze EQUAL-WIDTH und EQUAL-FREQUENCY ohne Rückgriff auf eine Klassifikation arbeitet.

Es konnte in Abschnitt 4.4 gezeigt werden, daß NUM_INT wie gefordert die effektive Bearbeitung großer Datenmengen erlaubt. Dabei kann NUM_INT als Vorteil für sich verbuchen, daß es im Gegensatz zu den anderen genannten eine vollständige Intervallhierarchie bildet, die mehr Information trägt als eine bloße Partitionierung.

Entsprechend der Warnung in [FPSSU96, Kap. 1], sich vor *perfekten KDD-Programmen* zu hüten, die jedes erdenkliche Problem lösen (Beware the Hype!), muß an dieser Stelle (leider) festgehalten werden, daß auch die in dieser Diplomarbeit vorgestellten Verfahren zu verbessern sind. Entsprechende Betrachtungen lassen sich in den Abschnitten 3.6 und 4.6 finden.

Im Hinblick auf das, was die Zukunft dem Gebiet des KDD bringen mag, ist von großem Interesse, ob die für die Motivation der Arbeit wichtige Einschätzung, daß es vorteilhaft sei, Verfahren direkt an eine Datenbank zu koppeln, (auch dann noch) Gültigkeit haben wird. Dabei sei insbesondere die

Technik des *Sampling* angesprochen, die einen anderen Weg beschreitet, mit großen Datenmengen zurecht zukommen: So wird dabei mit Hilfe statistischer Überlegungen versucht, einen Teil der Daten so geschickt auszuwählen, das sich die erzielten Ergebnisse ohne Probleme auf die gesamte Datenmenge übertragen lassen.

CLUST_DB und NUM_INT sind zum einen insofern eigenständig, als daß Einsatzmöglichkeiten aufgezeigt wurden, in denen sie ohne weitere Verfahren *stand-alone* arbeiten können. Man denke etwa an die bereits erwähnte Dimensionsreduktion oder den in Abschnitt 3.3.4 beleuchteten Aspekt funktionale Abhängigkeit – Datenqualität.

Auf der anderen Seite ist die primäre Ausgabe der numerischen und symbolischen Wertehierarchien auf Lernverfahren der nächsthöheren Ebene angewiesen. Wie groß der Beitrag der vorgestellten Ansätze und Verfahren für den Bereich der Wissensentdeckung in Datenbanken ist, wird auch davon abhängen, wie erschöpfend und geschickt diese die von CLUST_DB und NUM_INT gewonnene Information nutzen.

Es bleibt an dieser Stelle zu hoffen übrig, daß die vorgestellten Verfahren CLUST_DB und NUM_INT in den „Werkzeugkasten“ derer aufgenommen werden, die sich mit der Wissensentdeckung in Datenbanken bzw. dem maschinellen Lernen befassen.

Anhang A

Experimente zur Dimensionsreduktion

Im folgenden wird die im Abschnitt 3.4.2 benutzte Auswertung von CLUST_DB bzgl. der einzelnen Quellattribute B_k abgedruckt. Dabei entspricht *mean* dem Mittelwert $\overline{s}_{B_k}$ und *disp.* der hier nicht relevanten Varianz $\sigma_{s_{B_k}}$ der Ähnlichkeitswerte.[1] Die (nachträglich) mit * gekennzeichneten Attribute wurden zur Dimensionsreduktion aus dem Datensatz entfernt.

A.1 Glass

```
 separability of source-attributes (0: high, 1: low):
 ----------------------------------------------------
 sim for RI:   mean: 0.014677   disp.: 0.000259
 sim for na:   mean: 0.044065   disp.: 0.002099
 sim for mg:   mean: 0.064849   disp.: 0.006324
*sim for al:   mean: 0.068351   disp.: 0.003045
 sim for si:   mean: 0.058594   disp.: 0.001951
*sim for k:    mean: 0.111387   disp.: 0.009098
 sim for ca:   mean: 0.040718   disp.: 0.002470
*sim for ba:   mean: 0.176298   disp.: 0.018932
*sim for fe:   mean: 0.165748   disp.: 0.007411
```

A.2 Iris

```
 separability of source-attributes (0: high, 1: low):
 ----------------------------------------------------
*sim for SEPAL_LENGTH:  mean: 0.716667   disp.: 0.007222
*sim for SEPAL_WIDTH:   mean: 0.716667   disp.: 0.007222
 sim for PETAL_LENGTH:  mean: 0.166667   disp.: 0.055556
 sim for PETAL_WIDTH:   mean: 0.250000   disp.: 0.041667
```

[1] Siehe hierzu auch Abschnitt 3.2.5.

A.3 Hepatitis

```
 separability of source-attributes (0: high, 1: low):
 ----------------------------------------------------
 sim for AGE:             mean: 0.200000   disp.: 0.000000
 sim for SEX:             mean: 0.500000   disp.: 0.000000
*sim for STEROID:         mean: 1.000000   disp.: 0.000000
*sim for ANTIVIRALS:      mean: 1.000000   disp.: 0.000000
*sim for FATIGUE:         mean: 1.000000   disp.: 0.000000
*sim for MALAISE:         mean: 1.000000   disp.: 0.000000
*sim for ANOREXIA:        mean: 1.000000   disp.: 0.000000
*sim for LIVER_BIG:       mean: 1.000000   disp.: 0.000000
*sim for LIVER_FIRM:      mean: 1.000000   disp.: 0.000000
*sim for SPLEEN_PALPABLE: mean: 1.000000   disp.: 0.000000
*sim for SPIDERS:         mean: 1.000000   disp.: 0.000000
*sim for ASCITES:         mean: 1.000000   disp.: 0.000000
*sim for VARICES:         mean: 1.000000   disp.: 0.000000
 sim for BILIRUBIN:       mean: 0.400000   disp.: 0.000000
 sim for ALK_PHOSPHATE:   mean: 0.400000   disp.: 0.000000
 sim for SGOT:            mean: 0.400000   disp.: 0.000000
 sim for ALBUMIN:         mean: 0.200000   disp.: 0.000000
 sim for PROTIME:         mean: 0.600000   disp.: 0.000000
*sim for HISTOLOGY:       mean: 1.000000   disp.: 0.000000
```

A.4 Labor-negotiations

```
 separability of source-attributes (0: high, 1: low):
 ----------------------------------------------------
*sim for DURATION:                        mean: 1.000000   disp.: 0.000000
 sim for WAGE_INCREASE_FIRST_YEAR:        mean: 0.600000   disp.: 0.000000
*sim for WAGE_INCREASE_SECOND_YEAR:       mean: 0.800000   disp.: 0.000000
*sim for WAGE_INCREASE_THIRD_YEAR:        mean: 0.800000   disp.: 0.000000
*sim for COST_OF_LIVING_ADJUSTMENT:       mean: 0.666667   disp.: 0.000000
 sim for WORKING_HOURS:                   mean: 0.600000   disp.: 0.000000
 sim for PENSION:                         mean: 0.666667   disp.: 0.000000
*sim for STANDBY_PAY:                     mean: 0.750000   disp.: 0.000000
 sim for SHIFT_DIFFERENTIAL:              mean: 0.600000   disp.: 0.000000
*sim for EDUCATION_ALLOWANCE:             mean: 1.000000   disp.: 0.000000
 sim for STATUTORY_HOLIDAYS:              mean: 0.600000   disp.: 0.000000
*sim for VACATION:                        mean: 1.000000   disp.: 0.000000
 sim for LONGTERM_DISABILITY_ASSISTANCE: mean: 0.500000   disp.: 0.000000
 sim for CONTRIBUTION_TO_DENTAL_PLAN:     mean: 0.666667   disp.: 0.000000
 sim for BEREAVEMENT_ASSISTANCE:          mean: 0.500000   disp.: 0.000000
 sim for CONTRIBUTION_TO_HEALTH_PLAN:     mean: 0.666667   disp.: 0.000000
```

Literaturverzeichnis

[Aha90] D. W. Aha. A study of instance-based algorithms for supervised learning tasks. Technical report, University of California, Irvine, CA, November 1990.

[AHU83] Alfred Aho, John Hopcroft, and Jeffrey Ullmann. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[AK95] P. Gancarski & Korczak A. Ketterlin. Conceptual clustering over one-to-many relationships. *Proceedings of the Workshop KDD (ECML-95)*, 1995.

[Bel95] Siegfried Bell. The expanded implication problem of data dependencies — preliminary version. Research Report 16, University of Dortmund, Lehrstuhl Informatik VIII, June 1995.

[BFOS84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[Bis91] Gilles Bisson. Learning of rule systems in a first order representation. Rapport de Recherche 628, LRI, University de Paris-Sud, 1991.

[Bro94] Peter Brockhausen. Entdeckung von funktionalen und unären Inklusionsabhängigkeiten in relationalen Datenbanken. Master's thesis, Universität Dortmund, August 1994. in german.

[Coo88] G. W. Milligan & M. C. Cooper. A study of standardization of variables in cluser analysis. *Journal of Classification*, 1988.

[CS96] Peter Cheeseman and John Stutz. Bayesian classification (autoclass): Theory and results. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI Press/The MIT Press, Menlo Park, California, 1996.

[Dat95] C.J. Date. *An Introduction to Database Systems*. The Systems Programming Series. Addison-Wesley Publishing Company, 6 edition, 1995.

[Dow94] C. S. Wallace & D. L. Dowe. Intrinsic classification by mml - the snob program. In *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*, 1994.

[Eve80] B. Everitt. *Cluster Analysis*. Heinemann Educational Books, Ltd., 1980.

[FB92] Hubert Feger and Paul De Boeck. Categories and concepts: introduction to data analysis. In Iven Van Mechelen, James Hampton, Ryszard S. Michalski, and Peter Theuns, editors, *Categories and Concepts. Theoretical Views and Inductive Data Analysis*, Cognitive Science Series, chapter 8, pages 173–202. Academic Press, London, 1992.

[FI93] Usama M. Fayyad and Keki B. Irani. Multi–interval discretization of continuous–valued attributes for classification learning. In Ruzena Bajcy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI 93*, pages 1022–1029, San Mateo, CA, 1993. Morgan Kaufmann.

[Fis36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 1936.

[Fis58] W. D. Fisher. On grouping for maximum homogenity. *American Statistic Association*, 1958.

[Fis87] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(Douglas H. Fisher):139–172, 1987.

[FPSSU96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/The MIT Press, Manlo Park, California, 1996.

[Gna88] R. Gnanadesikan. Panel on discriminant analysis, classification and clustering. In *Discriminant Analysis and Clustering*. National Academy Press, Washington, DC, 1988.

[Gow66] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 1966.

[Gow71] J. C. Gower. *Statistical methods of comparing different multivariate analyses of the same data*. University Press, Edinburgh, 1971.

[Hub92] P. Arabie & L. J. Hubert. Combinatorial data analysis. *Annual Review pf Psychology*, 1992.

[Ker88] R. Kerber. Using a generalization hierarchy to learn from examples. In *Fifth International Conference on Machine Learning*. Morgan Kaufmann, 1988.

[Ker92] R. Kerber. Chimerge: Discretization of numeric attributes. In *AAAI-92*. Morgan Kaufmann, 1992.

[KF51] J. Lukaszewicz et al. K. Florek. Sur la liason et la division des points d un ensemble fini. *Colloquium Mathematics*, 1951.

[Kop92] Helmut Kopka. *LATEX: Eine Einführung.* Addison–Wesley, 4th edition, 1992.

[KW92] J.-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task–oriented models. In Stephen Muggleton, editor, *Inductive Logic Programming.*, number 38 in The A.P.I.C. Series, chapter 16, pages 335–360. Academic Press, London [u.a.], 1992.

[Leb87] Michael Lebowitz. Experiments with incremental concept formation: UNIMEM. *Machine Learning*, 2:103–138, 1987.

[Lev70] I. Gitman & M. D. Levine. An algorithm for detecting unimodal fuzzy sets and its application as a clustering technique. *IEEE Trans. Comp.*, 1970.

[Lin94] Guido Lindner. Logikbasiertes Lernen in relationalen Datenbanken. Forschungsbericht 12, Universität Dortmund, Lehrstuhl Informatik VIII, September 1994.

[Mac67] J. MacQueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematics, Statistics and Probability*, 1967.

[MHMT92] Iven Van Mechelen, James Hapton, Ryszard S. Michalski, and Peter Theuns. *Categories and Concepts. Theoretical Views and Inductive Data Analysis.* Cognitive Science Series. Academic Press, London, 1992.

[Mic83] Ryszard S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning — An Artificial Intelligence Approach*, volume 1, chapter 4, pages 83–135. Morgan Kaufmann, Palo Alto, CA, 1983.

[Mor93] K. Morik. Maschinelles Lernen. In Günther Görz, editor, *Einführung in die Künstliche Intelligenz.* Addison Wesley, 1993.

[MS83] Ryszard S. Michalski and Robert E. Stepp. Learning from observation conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning — An Artificial Intelligence Approach*, volume 1, chapter 11, pages 331–365. Morgan Kaufmann, Palo Alto, CA, 1983.

[Mur92] Fionn D. Murtagh. Cluster analysis using proximities. In Iven Van Mechelen, James Hampton, Ryszard S. Michalski, and Peter Theuns, editors, *Categories and Concepts. Theoretical Views and Inductive Data Analysis*, Cognitive Science Series, chapter 9, pages 225–246. Academic Press, London, 1992.

[MWKE93] K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications.* Knowledge-Based Systems. Academic Press, London u.a., 1993.

[PA87] J. D. Caroll & W. S. DeSarbo P. Arabie. *Three-way Scaling and Clustering.* Sage, Newbury Park, Ca, 1987.

[Paz95] Michael J. Pazzani. An iterative improvement approach for the discretization of numeric attributes in Bayesian classifiers. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *The First International Conference on Knowledge Discovery and Data Mining*, pages 228–233. AAAI Press, 1995.

[PSF91] Gregory Piatetsky-Shapiro and William J. Frawley, editors. *Knowledge Discovery in Databases.* The AAAI Press, Menlo Park, 1991.

[Qui83] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning — An Artificial Intelligence Approach*, volume 1, chapter 15, pages 463–483. Morgan Kaufmann, Palo Alto, CA, 1983.

[Qui86] J. R. Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning – An Artificial Intelligence Approach*, volume 2, chapter 6, pages 149–166. Morgan Kaufmann, Palo Alto, CA, 1986.

[Qui93] John Ross Quinlan. *C4.5: Programs for Machine Learning.* Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[Ros95] M. Richeldi & M. Rossotto. Class-driven statistical discretization of continuous attributes. In *ECML-95*, 1995.

[Sal91] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 1991.

[Sne69] J. W. Carmichael & P. H. A. Sneath. Taxometric maps. *Systematic Zoology*, 1969.

[Sok73] P. H. A. Sneath & R. R. Sokal. *Numerical Taxonomy.* Freeman and Co., San Francisco, 1973.

[Ull88] Jeffrey D. Ullman. *Principles of database and knowledge–base systems*, volume 1. Computer Science Press, Rockville, MD, 1988.

[WD95] Dietrich Wettscherek and Thomas G. Dietterich. An experimental comparison of the nearest-neighbour and nearest-hyperrectangle algorithms. *Machine Learning*, 19(1):5 – 27, April 1995.

[Wes95] Stefanie Wessel. Lernen qualitativer Merkmale aus numerischen Robotersensordaten. Master's thesis, Universität Dortmund, 1995.

[Wet94] D. Wettschereck. *A study of Distance-Based Machine Learning Algorithms*. PhD thesis, Oregon State University, 1994.

[Wil62] S. Wilks. *Mathematical Statistics*. Wiley and Sons, New York, 1962.

[Wil66] G. N. Lance & W. T. Williams. Computer programs for hierarchical polythetic classification. *Computer Journal*, 1966.

[Wil67] G. N. Lance & W. T. Williams. A general theory of sorting strategies: 1 hierarchical systems. *Computer Journal*, 1967.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 1965.

Abbildungsverzeichnis

Tabellenverzeichnis