JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

KOMBINIERTES DATA MINING – KLASSIFIKATION UNTER VERWENDUNG VON DURCH CLUSTERING GEWONNENEN HILFSINFORMATIONEN

DIPLOMARBEIT

zur Erlangung des akademischen Grades „Mag.rer.soc.oec."
eingereicht von

MARKUS HUMER

am Institut für Wirtschaftsinformatik
der Johannes-Kepler-Universität Linz
Abteilung für Data & Knowledge Engineering

Begutachter: o. Univ. Prof. Dr. Michael Schrefl
Betreuer: Dipl.-Wirtsch.-Inf. Mathias Goller

Linz, Oktober 2004

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Linz, im Oktober 2004 .....

Abstract

Die Aufgabenstellungen im Data Mining sind sehr vielseitig und können in manchen Fällen nur durch eine kombinierte Anwendung verschiedener Data-Mining-Verfahren gelöst werden. Zwar existieren bereits Ansätze, in denen unterschiedliche Verfahren hintereinander ausgeführt werden, ihre Anwendung wird jedoch unabhängig voneinander betrachtet.
Diese Arbeit vertieft als Weiterführung einer parallel durchgeführten Studie [SK04] den Begriff des „Kombinierten Data Mining“. Beim Kombinierten Data Mining interagieren die angewendeten Methoden miteinander. Durch diese Interaktion soll sich hinsichtlich der Qualität und/oder Effizienz eine Verbesserung gegenüber einer unabhängigen Ausführung einstellen. Dies kann z.B. durch die Weitergabe von Hilfsinformationen des vorgelagerten Verfahrens an das nachfolgende Verfahren erreicht werden.
Im Zuge dieser Arbeit werden ein Clustering- und ein Klassifikations-Algorithmus miteinander kombiniert. Dazu wird ein Entscheidungsbaum-Klassifikator implementiert, der Hilfsinformationen, die in einem vorher ausgeführten Clustering identifiziert und berechnet wurden, einfließen lässt. Es soll untersucht werden, welche Hilfsinformationen sich für die Konstruktion des Klassifikators eignen und welchen Einfluss sie auf die Qualität des Klassifikators haben.

Problems in data mining are versatile. In some cases they can only be solved by combining different data mining methods. Existing approaches concerning a combined use of data mining methods consider that topic under an isolated point of view.
This work continues a parallel study [SK04] and further introduces the term “Combined Data Mining”. Through combination of data mining methods the result of the combined process should gain quality and efficiency by letting the used methods interact with each other. One possible way to achieve interaction is to compute additional information in a preliminary step which is used in a succeeding step.
In the course of this work clustering and classification become combined. Therefore a “decision tree classifier” is implemented, which uses by a clustering algorithm previously identified and computed additional information. Priority objective is to investigate additional information that can simply be applied to the classifier and has impact on the quality of the classifier.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das letzte Jahrzehnt war geprägt von einem explosionsartigen Anstieg unserer Fähigkeiten Daten zu generieren und zu sammeln. Fortschritte in der Datensammlung, der Einsatz von Chipkarten und die Umstellung vieler betriebswirtschaftlicher Transaktionen auf computerunterstützte Datenverarbeitung führten zu einer Flut von neuen Informationen. Daraus resultierend war und ist es notwendig neue Techniken und Werkzeuge zu entwickeln und einzusetzen, die es ermöglichen, Daten intelligent und automatisch in nutzbares Wissen umzuwandeln.

Traditionelle Methoden der Datenanalyse, d.h. dass der Mensch direkt mit den Daten interagiert, bieten wenig Spielraum, große Datenbestände effizient zu bearbeiten. Man ist nur bedingt in der Lage große Datenbestände strukturiert und effizient verwalten und speichern zu können bzw. schnell und unkompliziert auf die abgelegten Daten zuzugreifen. Als Methoden, Daten zu definieren, zu manipulieren und aus der Datenbank zu filtern, aufzubereiten und auszugeben, stehen relationale Datenbanksprachen wie SQL zur Verfügung. Abfragen über Datenbanksprachen liefern **„Daten im engeren Sinne“**, d.h. sie liefern die Daten selbst in vordefinierten Strukturen oder Ergebnisse aus Berechnungen mithilfe dieser Daten.
Die Möglichkeit, den Anwender beim Verstehen und Analysieren großer Datenbestände zu unterstützen, ist jedoch mit diesen Methoden nicht gegeben [FPSU].

Mit den Methoden des Data Mining versucht man diese Lücke zu schließen. Data Mining wird zur automatischen **Extraktion von Mustern** aus großen Datenbeständen, die Wissen beinhalten, das implizit in den Daten vorhanden ist, eingesetzt [HK01]. Man versucht zwischen den Zeilen der Daten zu lesen, um **„Muster“** zu erhalten, die mittels Evaluationsverfahren in Wissen umgewandelt werden können.
Data Mining und „Knowledge Discovery in Databases“ (KDD) werden häufig als Synonyme verwendet. Der grundlegende Unterschied zwischen Data Mining und KDD liegt darin, dass Data Mining den Kern des gesamten KDD-Prozesses bildet [FR02].

Jene zwei großen Forschungsgebiete der Informatik, die maßgeblich zur Entwicklung von Data Mining beigetragen haben, sind der Einsatz von (großen) Datenbanken und die „Künstliche Intelligenz“ (KI) [SM00]. KI bzw. „maschinelles Lernen“ beschäftigen sich damit, wie Menschen und Maschinen (Algorithmen) von Daten lernen können. Der Forschungsbereich der KI versucht Algorithmen zu konstruieren, die Informationen in einer Form speichern und wiedergeben können, die dem menschlichen Lernen sehr ähnlich ist. Die Forschung auf diesem Gebiet ging weg von Modellierungen, die darauf abzielten, wie Menschen lernen, hin zur Entwicklung von Algorithmen, die für spezielle Probleme trainiert (angelernt) und verwendet wurden (z.B. Prognosen von Klassenzugehörigkeiten). Dies führte zu einer Überschneidung mit den Methoden der angewandten Statistik mit großer Gewichtung auf Klassifikations-Techniken wie Entscheidungsbäume bzw. Entscheidungsregeln. Die Methodik und Ergebnisse dieser Techniken konnten auch von Personen leicht verstanden werden, die sich nicht professionell mit Data Mining auseinandersetzen.

Die Anwendungsgebiete von KDD und Data Mining reichen vom Marketing (Marktsegmentierung, Warenkorbanalysen, zielgerichtetes Marketing etc.), über Risikomanagement (Qualitätskontrolle, Konkurrenzanalyse etc.) bis hin zur Betrugserkennung [HK01].
Je Nach Anwendungsgebiet kann es zielführend sein, genau ein einziges oder mehrere Data-Mining-Verfahren zu verwenden, um eine Aufgabenstellung zu lösen. Ist die Aufgabe des Data Mining so gestellt, dass man zwei oder mehrere Data-Mining-Verfahren gemeinsam auf den vorhandenen Datenbestand anwenden muss, so ist es notwendig, Data-Mining-Methoden hintereinander (sequentiell) einzusetzen. Das Hintereinanderausführen von verfügbaren Methoden wird mit dem Schlagwort „Kombiniertes Data Mining“ in Beziehung gesetzt. Ziel dabei ist es, durch den wiederholten Prozess der Datenextraktion, effizientere, d.h. leichter interpretierbare, und qualitativ hochwertigere, d.h. fehlerreduzierte, Ergebnisse zu erhalten. Wie dies im Detail geschieht, wird in späteren Kapiteln genauer erläutert.

Diese Arbeit beschäftigt sich mit dem Versuch der Steigerung der Qualität des Ergebnisses bei wiederholter kombinierter Anwendung von verschiedenen Data Mining Verfahren. Die Steigerung der Performanz (Laufzeit) ist ein positiver

Nebeneffekt der kombinierten Anwendung zweier Verfahren. Die Qualitätssteigerung ist jedoch vorrangiges Ziel dieser Arbeit.
Zur Erreichung dieses Ziels wird vor allem die Kombination von Clusteringverfahren mit Klassifikationsverfahren sowohl praktisch als auch theoretisch untersucht.
Der Begriff „Kombiniertes Data Mining“ definiert sich laut [GM04] wie folgt:

Definition: Beim „Kombinierten Data Mining“ wird ein Data Mining Verfahren *A* vor einem Data Mining Verfahren *B* ausgeführt, sodass *B* von *A* profitiert. *B* kann dazu das Ergebnis von *A* oder/und eigens ermittelte Hilfsinformationen von *A* für *B* nutzen. *B* profitiert dann von *A* wenn das Ergebnis von *B*, gemäß einem geeigneten Gütemaß, besser ist, oder/und sich die Laufzeit von *B* verringert.

Des Weiteren besteht die Möglichkeit eines parallelen Einsatzes von Data-Mining-Verfahren. Hierbei verwendet ein Data-Mining-Verfahren die Methoden und Techniken eines anderen Verfahrens während der Ergebnisfindung.

1.1 Problemstellung

Kombiniertes Data Mining beschäftigt sich also mit der Steigerung der Effizienz und der Qualität der Ergebnisse bei wiederholter Anwendung von verschiedenen Data-Mining-Methoden.
Clusteringverfahren zum Beispiel gruppieren Daten anhand Ähnlichkeiten in den untersuchten Merkmalen von Datensätzen. Ob das gefundene Ergebnis tatsächlich eine befriedigende Lösung der ursprünglichen Aufgabenstellung darstellt, kann der Algorithmus nicht feststellen, da er die Problemstellung nicht kennt, bzw. nicht begreifen kann, da Clustering-Algorithmen Daten ohne Vorwissen in Klassen (Cluster) einzuteilen versuchen.
Das bedeutet, dass die Ergebnisse durch den Menschen beurteilt werden müssen. Aufgrund der Clusterdaten kann dieser das aber nicht tun. Erst die Klassifikation von den geclusterten Daten liefert dem Anwender die nötigen beschreibenden Daten, um die Güte des Clustering beurteilen zu können. Die naive Anwendung von Klassifikation zur Beschreibung der Qualität von Clusteringergebnissen ist aus Effizienzsicht suboptimal. Data Mining befasst sich definitionsgemäß mit sehr großen

Datenbeständen. Demzufolge ist der Ressourcenverbrauch an Rechenzeit und Speicher nicht vernachlässigbar.
Eine Möglichkeit der Reduzierung von Ressourcenbedarf ist die Berechnung von Hilfsinformationen im ersten Schritt für die folgenden Schritte. Eine weitere Möglichkeit zur Reduktion besteht in der Kenntnis der verwendeten Verfahren. Die Meta-Information über die Vorgänger-Verfahren erlaubt dem nachfolgenden Verfahren von einem vereinfachten Problem auszugehen.

1.2 Ziel und Zweck der Arbeit

Ziel dieser Diplomarbeit ist die Ermittlung jener Hilfsinformationen, die besonders nützlich für die weitere Klassifizierung sind. Hilfsinformationen sind genau dann nützlich, wenn sie einen positiven Einfluss auf das Ergebnis der Klassifikation nehmen, d.h. z.B. dass die Anzahl der richtig klassifizierten Objekte ansteigt.
Dazu wird ein Klassifikationsverfahren als Programm umgesetzt, ein „Decision Tree Classifier“, der so implementiert wird, dass Schnittstellen zur Verfügung stehen, die es ermöglichen, verschiedene Kombinationen von Hilfsinformationen in den Algorithmus einfließen zu lassen.
Als Input für Training und Tests dienen jene Hilfsinformationen, die in einer parallel durchgeführten Studie [SK04] identifiziert wurden. Die in die Klassifikations-Verfahren einfließenden Meta-Informationen stammen ausschließlich von genau einem Clusteringverfahren. Dieses Verfahren wurde in [SK04] entwickelt und verwendet ein K-Means-Clustering zur Berechnung von Hilfsinformationen. Alleine diese Einschränkung kann als Metainformation im nachfolgenden Klassifikations-Verfahren verwertet werden. Das K-Means-Clustering erzeugt konvexe Cluster. Die Form der Cluster lässt es zu, zielgerichtet Punkte auszuwählen, um einzelne Qualitätskriterien des Klassifikators gezielt zu beeinflussen. Aus diesem Grund liefert das Vorgängerverfahren als zusätzliche Ergebnisse Punkte, die den Clustermittelpunkten möglichst nah bzw. möglichst fern sind. Dabei beeinflussen z.B. nahe Punkte die Klassifikationsgenauigkeit, während es entfernte Punkte ermöglichen, Punkte zu klassifizieren, die den Klassen schwer zuzuordnen sind. Abbildung 1 soll den Zusammenhang zwischen Vorgänger- und Nachfolgeralgorithmus veranschaulichen:

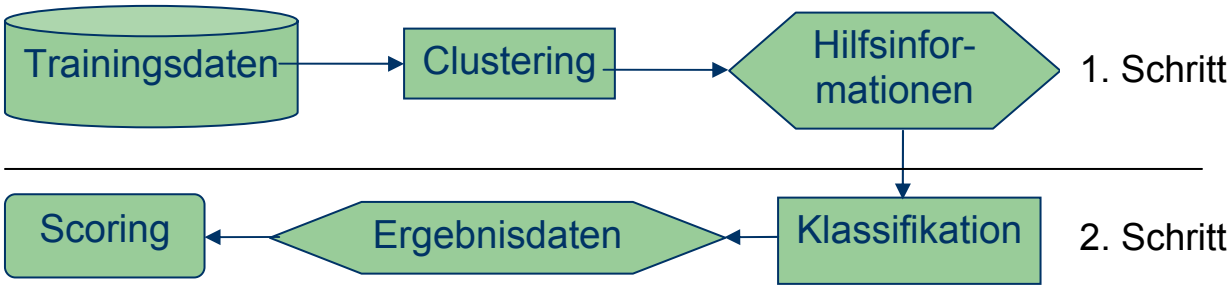


Abbildung 1: Ablaufdiagamm „Kombiniertes Data Mining“ mit Clustering und Klassifikation

Die Tests sollen Aufschluss darüber geben, welchen Qualitätszuwachs bzw. Qualitätsverlust man in den Endergebnissen des Klassifikationsverfahrens messen kann, wenn man die identifizierten Hilfsinformationen bei der Generierung der Klassifikatoren berücksichtigt. Bei der Klassifikation ist hauptsächlich die Qualität entscheidend. Die Laufzeit von Klassifikationsalgorithmen bewegt sich meist in einem tolerierbaren Ausmaß.
Das Ergebnis der Diplomarbeit soll alle identifizierten Hilfsinformationen auflisten und nach ihrer Brauchbarkeit in Bezug auf die Qualität des Endergebnisses des Klassifikations-Algorithmus (Klassifikator) beurteilen. Darüber hinaus soll eine etwaige Steigerung der Effizienz einer kombinierten Anwendung bestimmt und dokumentiert werden.

1.3 Aufbau der Arbeit

Die weiteren Kapitel dieser Arbeit gliedern sich wie folgt:

Kapitel 2 beschäftigt sich mit den Grundlagen des Knowledge Discovery in Databases und den dabei verwendeten Data-Mining-Verfahren. Es wird ein kurzer Überblick über den KDD-Prozess gegeben und die Data-Mining-Verfahren des Clusterings und der Klassifikation vorgestellt. Auf die Methoden der Klassifikation wird im Besonderen eingegangen, um sie einander gegenüberzustellen und das Themengebiet dieser Arbeit möglichst genau abzugrenzen.

Kapitel 3 zeigt gängige Verfahren zur Bewertung von Klassifikatoren und diskutiert Kriterien an denen die Qualität von Klassifikatoren gemessen werden kann. Des

Weiteren wird unter dem Aspekt der Qualitätsverbesserung das Thema „Fehlerreduktionspruning“ angeschnitten.

Kapitel 4 definiert den Begriff „Kombiniertes Data Mining“ und es werden drei Vorgehensweisen des Kombinierten Data Mining vorgestellt. Es wird auf ähnliche bzw. verwandte Verfahren Bezug genommen und eine Einordnung dieser Arbeit in einen der vorgestellten Archetypen von kombiniertem Data Mining vorgenommen.

Kapitel 5 befasst sich mit den Typen von Hilfsinformationen, die den Klassifikationsalgorithmus bei der Generierung des Klassifikators unterstützen können. Es werden jene Hilfsinformationen beschrieben, die im Vorgängeralgorithmus (Clustering durch Varianzminimierung – „K-Means“) zur Laufzeit identifiziert wurden und in weiterer Folge deren Potential hinsichtlich einer Qualitätssteigerung des Nachfolgeralgorithmus (Decision Tree Classifier – RainForest-Framework) geschätzt.

Kapitel 6 beschäftigt sich mit den Algorithmen, die im Zuge des kombinierten Data-Mining-Prozesses verwendet wurden. Das die Trainings- und Testdaten liefernde Clusteringverfahren wird in seinen Grundzügen besprochen und das Verfahren zur Generierung des Entscheidungsbaum-Klassifikators wird im Detail erläutert.

Kapitel 7 befasst sich mit der Implementierung des Decision-Tree-Algorithmus und dessen Dokumentation. Die Dokumentation beschreibt einerseits das Training des Entscheidungsbaumes und die mittels der Hilfsinformationen eingefügten Modifikationen, andererseits die im Programm umgesetzten Testmethoden bzw. die Routinen zur Ermittlung der Klassenzugehörigkeiten und der Qualität der generierten Klassifikatoren.

Kapitel 8 beschreibt Trainings- und Testdaten bzw. die auf den implementierten Decision-Tree-Algorithmus angewandten Testreihen. Die in Kapitel 3 eingeführten und in den Tests ermittelten Qualitätskriterien werden in tabellarischer und grafischer Form dargestellt. Abschießend werden die verwendeten Hilfsinformationen einander hinsichtlich ihres Einflusses auf die Klassifikatorqualität und ihrer daraus resultierenden Brauchbarkeit gegenübergestellt.

Kapitel 9 fasst die Arbeit in ihren wichtigsten Punkten zusammen und zieht ein Fazit über die gewonnen Erkenntnisse.

2 Grundlagen

Nachdem heutzutage riesige Mengen an Daten in Datenbanken abgelegt sind und diese Tendenz stetig steigt [FR02], besteht die Notwendigkeit der Entwicklung und des Einsatzes von (semi-)automatischen Methoden zur Extraktion von implizitem Wissen. Die erfolgreiche Entdeckung solch versteckter Informationen in Datenbanken, kann in Unternehmen durch die Verbesserung von Entscheidungsprozessen verwertet werden. Daten in Verkaufsdatenbanken können beispielsweise Informationen über das Kaufverhalten von Kunden bezüglich bestimmter Güter enthalten. Auch in den Patientendaten von Krankenhäusern kann Wissen enthalten sein, dass dazu verwendet werden kann, für zukünftige Patienten bessere Diagnosen zu stellen und Behandlungen vorzunehmen.

Das nachfolgende Unterkapitel definiert den Begriff „Knowledge Discovery in Databases“ (KDD) und führt in dessen Grundlagen ein. Des Weiteren werden Methoden des Data Mining vorgestellt und eine Einordnung des Data Mining in den sog. KDD-Prozess vorgenommen. Die darauf folgenden Kapitel geben Einblick in die Data-Mining-Verfahren des Clusterings und der Klassifikation, wobei für beide Verfahren die wichtigsten Methoden genauer erläutert werden.

2.1 Knowledge Discovery in Databases

Data Mining und „Knowledge Discovery in Databases“ (KDD) werden in der Praxis oft als Synonyme für ein und denselben Prozess verwendet [FR02]. Eine Unterscheidung dieser beiden Begriffe wurde in [FPS96] gegeben. Data Mining ist das Kernstück des komplexeren KDD-Prozesses (vgl. Abbildung 2). KDD beinhaltet zusätzlich Schritte der Datenaufbereitung und der nachträglichen Validierung der Ergebnisse. Im weiteren Verlauf dieser Arbeit werden die Begriffe Data Mining und KDD gleichbedeutend verwendet.
[ES00] definieren KDD folgendermaßen: „Knowledge Discovery in Databases“ beschäftigt sich mit Methoden und Prozessen zur (semi-)automatischen Extraktion von Wissen aus Datenbanken. Dieses Wissen soll

- gültig (im statistischen Sinne)
- bisher unbekannt und
- potentiell nützlich sein.

Data Mining umfasst somit den Prozess der Gewinnung neuer, valider und handlungsrelevanter Informationen aus großen Datenbanken und die Nutzung dieser Informationen für betriebswirtschaftliche Entscheidungen.

Die Methoden des Data Mining sind domänenübergreifende Datenanalysemethoden aus Statistik, Künstlicher Intelligenz, Maschinellem Lernen und Mustererkennung zur Auswertung großer Datenbestände. Diese stehen zwar im Fokus dieses Prozesses, generieren aber ohne zielorientiertes Vorbereiten der Ausgangsdaten und Evaluieren der Ergebnisdaten oft keine oder sogar irreführende Informationen [FA96].

Daher ist es zweckmäßig KDD als iterativen Prozess zu verstehen.
Lediglich 10 % des Zeitaufwandes im Data-Mining-Prozess entfallen unmittelbar auf den Einsatz von Data-Mining-Methoden, 90 % fließen in die Datenaufbereitung und Ergebnisnachbearbeitung [CA97].

Der Prozess des KDD gliedert sich nach [FPS96] wie folgt:

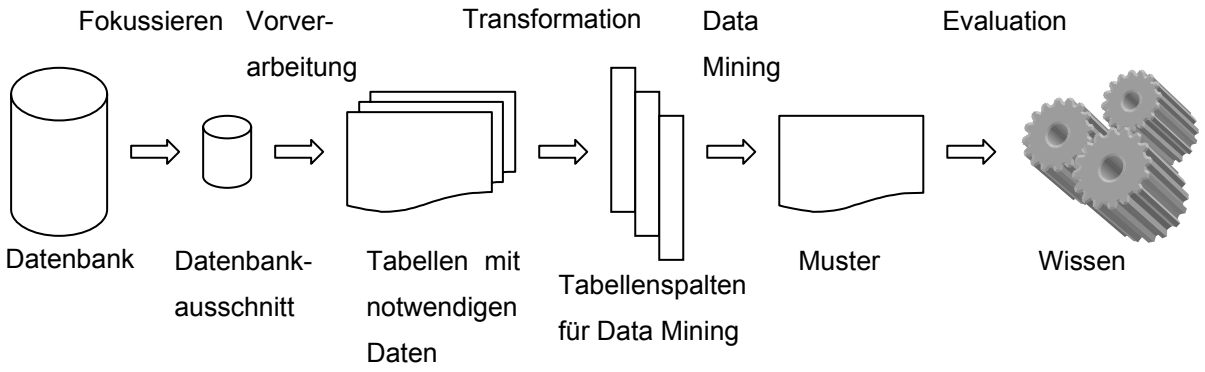


Abbildung 2: Der KDD-Prozess

Fokussieren

In dieser Prozessphase wird versucht, das Verständnis für die durchzuführende Anwendung zu erlangen und zu vertiefen. Es werden Ziele des KDD bezüglich der

gegebenen Anwendung definiert. Darüber hinaus wird geklärt, wie die Daten gehalten und verwaltet werden sollen, um Redundanzen und Inkonsistenzen vorzubeugen (Integration des KDD mit kommerziellen Datenbanksystemen).

Vorverarbeitung

In der Vorverarbeitungsphase werden alle für das eigentliche Data Mining benötigten Daten integriert, konsistent gemacht und vervollständigt. Diese Phase nimmt einen großen Teil des Gesamtaufwandes ein. Durch die Integration werden Daten, die nach unterschiedlichen Konventionen (z.B. Spaltennamen) gehalten werden, zusammengeführt. Inkonsistenzen betreffen vor allem unterschiedliche Werte desselben Attributs oder Schreibfehler. Es kann sich jedoch auch um Rauschen – ein zufällig auftretendes Muster – handeln, welches in dieser Phase entfernt werden sollte.

Transformation

Sind die Ausgangsdaten integriert, konsistent und vollständig werden sie in eine für das Ziel des KDD geeignete Form transformiert. Es werden z.B. bestimmte Attribute zur Weiterverarbeitung selektiert oder Attributwerte diskretisiert.

Data Mining

Hier geschieht die eigentliche Anwendung von Data-Mining-Algorithmen, die im vorbereiteten Datenbestand nach gültigen Mustern suchen. Diese Algorithmen lassen sich nach [ES00] folgenden aufgabenbezogenen Verfahren zuordnen:

- **Clustering**

  Beim Clustering versucht man eine unklassifizierte Menge von Daten anhand von Ähnlichkeiten in Gruppen (Clustern) zusammenzufassen und von unähnlichen Objekten möglichst abzugrenzen. Nicht zugeteilte Objekte werden als Ausreißer bezeichnet.

- **Klassifikation**

  Bei der Klassifikation wird versucht, eine vorklassifizierte Menge von Daten (die sog. Trainingsmenge) so zu verwerten, dass ein Klassifikator „gelernt“ wird, der neue/zukünftige Objekte anhand eines Vergleichs ihrer

Attributausprägungen mit dem Klassifikator einer vorgegebenen Klasse zuweist.

- **Assoziationsregeln**
  Aufgabe hierbei ist es Regeln zu definieren, die häufig auftretende und starke Verbindungen innerhalb von Datenbanktransaktionen beschreiben (z.B. WENN Gut1 gekauft wird DANN wird auch Gut2 gekauft).

- **Generalisierung**
  Bei der Generalisierung werden Attributwerte generalisiert und die Anzahl der Datensätze reduziert, um eine möglichst kompakte Repräsentation der Daten zu erhalten.

Evaluation

Die abschließende Phase des KDD-Prozesses besteht darin, die gefundenen Muster in geeigneter Art zu präsentieren. Sollten die von einem Experten evaluierten Ergebnisse nicht den in der Phase „Fokussieren“ definierten Zielen entsprechen, muss unter Umständen der gesamte KDD-Prozess oder ein Teil des Prozesses, zum Beispiel der Data Mining Schritt, mit abgeänderten Parametern solange wiederholt werden bis die Evaluation erfolgreich ist. Erst dann wird das gewonnene Wissen in das bestehende System integriert und kann für neue KDD-Prozesse genutzt werden.
Für eine erfolgreiche Evaluierung der Ergebnisse ist die Präsentation der gefundenen Muster durch das System sehr entscheidend. Dies ist vor allem dann schwierig, wenn entweder sehr viele Muster gefunden werden (z.B. Identifikation von Assoziationsregeln) oder die Zahl der verwendeten Attribute sehr groß ist. Daher ist eine Visualisierung der Ergebnisse vorzuziehen, da sie für den Benutzer fast immer leichter interpretierbar ist als eine textuelle Darstellung. Entscheidungsbäume (Kapitel 2.3.3), die den Hauptgegenstand dieser Arbeit bilden, eignen sich besonders gut zur visuellen Darstellung [ES00].

In den folgenden Unterkapiteln werden die Data Mining Verfahren Clustering und Klassifikation beschrieben. Dabei wird bei der Klassifikation besonders auf die Entscheidungsbaum-Klassifikation eingegangen. Die beiden Verfahren werden in

weiterer Folge auch als überwachtes Lernen (das „Lernen“ eines Modells bei der Klassifikation) und unüberwachtes Lernen (oder Clustering) bezeichnet [HK00].

2.2 Clustering

Der Prozess des Gruppierens einer Menge von physischen oder abstrakten Objekten in Klassen von gleichartigen Objekten wird „Clustering“ genannt [HK00]. Die so genannte Clusteranalyse ist eine sehr wichtige bereits in der Kindheit des Menschen angewendete Aktivität. So wie ein kleines Kind durch unterbewusste Cluster-Schemata zwischen Katzen und Hunden bzw. Tieren und Pflanzen zu unterscheiden lernt, werden auch in der Wissenschaft viele Forschungsbereiche wie z.B. Muster-Erkennung, Bildverarbeitung oder Marktforschung durch die Clusteranalyse unterstützt.

Nach [ES00] ist das Ziel von Clusteringverfahren folgendermaßen definiert:

- Daten (semi-)automatisch
- so in Kategorien, Klassen oder Gruppen (Cluster) einzuteilen, dass
- Objekte im gleichen Cluster möglichst ähnlich
- und Objekte aus verschiedenen Clustern möglichst unähnlich zueinander sind.

Zu den Einsatzcharakteristika des Clusterings zählt, dass das Verfahren keine vorklassifizierten Daten benötigt und zunächst kein Zielattribut (wie z.B. „Versicherung? ja/nein“ bei der Entscheidungsbaum-Klassifikation) benötigt wird. Das Primärziel besteht darin, Ähnlichkeit zwischen Objekten zu identifizieren und nicht Abhängigkeiten zwischen Merkmalen zu entdecken. Diese Ähnlichkeiten werden für Wertepaare (Punkte) mittels Distanzfunktionen bestimmt. Eine große Distanz zwischen zwei Punkten bedeutet, dass es sich um unähnliche Wertepaare handelt, während eine kleine Distanz darauf schließen lässt, dass sich die Wertepaare ähnlich sind. Zur Berechnung einer solchen Distanz existiert eine Reihe von Distanzfunktionen. Nachfolgend sind zwei häufig verwendete Funktionen für die Bestimmung der Distanz zwischen zwei Punkten x und y mit numerischen Attributsausprägungen $x = (x_1, \ldots, x_n)$ und $y = (y_1, \ldots, y_n)$ erwähnt [ES00]:

- **Euklidische Distanz:** $dist(x, y) = \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2}$
- **Manhattan Distanz:** $dist(x, y) = |x_1 - y_1| + ... + |x_n - y_n|$

Clusteringverfahren sind in der Lage Daten ohne das Wissen über ihre Klassenzugehörigkeit zu gruppieren. Aus diesem Grund wird das Clustering häufig anderen Data-Mining-Verfahren (z.B. Klassifikation) vorgeschaltet.
Die Grundschritte beim Durchführen eines Clusterings sind in Abbildung 3 dargestellt:

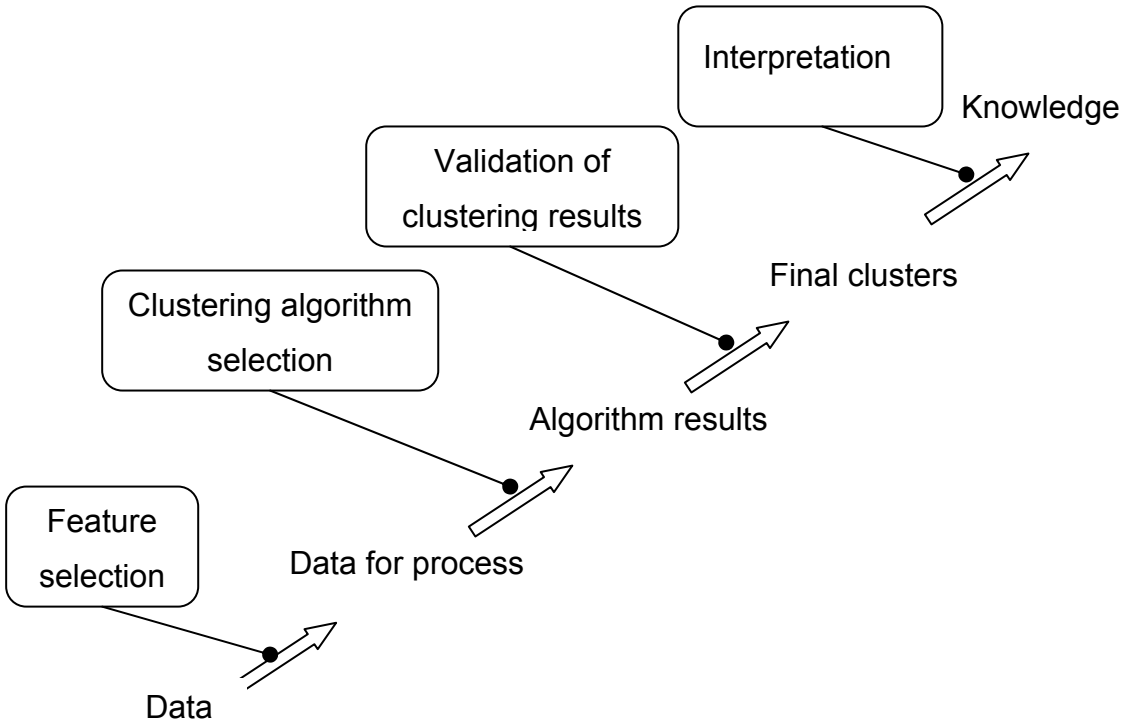


Abbildung 3: Phasen des Clusteringprozesses nach [VHG03]

- **Attributauswahl (Feature selection)**
  Ziel hierbei ist es, jene Daten und Merkmale, die für das Clustering von Interesse sind, zu selektieren und jegliche Information, welche die Aufgabe des Data-Mining-Verfahrens unterstützen kann, zu sammeln. Die Auswahl der Merkmale sollte besonders gründlich erfolgen, denn

  - zu viele Merkmale können sich negativ auf die Fehlerrate auswirken ("curse of dimensionality").

- der Speicherbedarf / die Rechenzeit nimmt mit der Anzahl der Merkmale stark zu.

- **Clusteringalgorithmus (Clustering Algorithm)**
  Es muss ein der Aufgabe gewachsener Algorithmus ausgewählt werden

- **Evaluierung (Validation of the results)**
  Nachdem Clustering-Algorithmen Klassen definieren, die nicht a priori bekannt sind, müssen die Zuordnungen a posteriori evaluiert werden.
  Dazu gibt es Maße für die Homogenität einer Klasse [ES00]:

  - z.B. mittleres Ähnlichkeitsmaß für alle Paarungen innerhalb der Klasse

  - mittlerer Abstand von einem Repräsentanten (Medoid) oder Schwerpunkt (Centroid) der Klasse

  Außerdem existieren Maße für die Güte des gesamten Clustering:

  - Varianzkriterium: $TD^2(C) = \sum_{\vec{x}^p \in C} \sum_{j=1}^{k} (\vec{x}^p{}_j - \bar{x}_j(C))^2 \rightarrow \min$

    - C: Cluster, C $\subset \mathbb{R}^k$
    - TD²: Kompaktheit eines Clusters (je kleiner der Wert für TD² ist, desto näher liegen die Objekte des Clusters beieinander, d.h. um so kompakter ist der Cluster)
    - $\vec{x}^p$: Objekte in einem k-dimensionalen euklidischen Vektorraum, $\vec{x}^p = \begin{pmatrix} x_1^p \\ ... \\ x_k^p \end{pmatrix}$, $\vec{x}^p \in \mathbb{R}^k$
    - $\vec{x}_j$: Merkmalsvektor eines Objekts im Cluster,
    - $\bar{x}_j$: Arithmetisches Mittel aller zu $\vec{x}^p$ gehörigen Merkmalsvektoren

  - Summe der Gütemaße der einzelnen Klassen

- **Interpretation (Interpretation of the results)**
  Die Ergebnisse sollten von Experten interpretiert werden, indem sie mit bereits vorhandenen oder experimentellen Daten zusammengeführt werden.

Man unterscheidet fünf Arten von Clusteringverfahren [VHG03]. Die beiden gängigsten Verfahren sind das partitionierende und das hierarchische Clustering. Bei der Auswahl und Anwendung eines Clusteringverfahrens sollte berücksichtigt werden, dass sich Cluster in Größe, Form und Dichte stark voneinander unterscheiden können bzw. auch ineinander verschachtelt liegen und Hierarchien bilden können [ES00]. Abbildung 4 zeigt Beispiele für 2-dimensionale Clusterstrukturen mit verschiedenen Charakteristika.

Abbildung 4: Beispiele für 2-dimensionale Clusterstrukturen [ES00]

Die zwei nun folgenden Unterkapitel erläutern kurz die wichtigsten partitionierenden und hierarchischen Clusteringverfahren und stellen sie einander vergleichend gegenüber.

2.2.1 Partitionierende Clusteringverfahren

Partitionierende Clusteringverfahren zerlegen die gegebenen Daten von n Datenobjekten in k Cluster, wobei jeder Cluster k eine Punktmenge kleiner gleich n beinhaltet. Dabei müssen folgende Bedingungen erfüllt sein:

- Jeder Cluster enthält mindestens ein Datenobjekt
- Jedes Datenobjekt gehört zu maximal einem Cluster

In dieser Kategorie von Clusteringverfahren ist **K-Means** [MQ67] ein allgemein verwendeter Algorithmus. Das Ziel von K-Means liegt in der Minimierung des Abstandes eines jeden Objektes im Cluster vom Clustermittelpunkt (Centroid). Dazu werden vor der Ausführung des Algorithmus zufällig Clusterzentren bestimmt. In weiterer Folge werden alle Objekte dem ihnen nächst gelegenen Centroid zugeordnet und daraufhin die Clustermittelpunkte neu berechnet. Diese Vorgehensweise wird solange wiederholt bis sich nach der Zuweisung der Objekte kein Centroid mehr verändert (vgl. auch Kap. 5.1).

Ein weiterer Algorithmus dieser Kategorie ist **PAM** (Partitioning Around Medoids). Im Unterschied zum bei K-Means verwendeten Centroid, der ein „virtuelles“ Objekt (arithmetisches Mittel aller Dimensionen) darstellt, wird bei PAM ein sog. Medoid eingesetzt. Der Medoid ist ein repräsentatives, echtes Objekt eines Clusters.
Der Algorithmus beginnt mit der Selektion von Medoiden für jeden Cluster und weist die Objekte dem Medoiden zu, dem sie am nächsten sind. Dann werden die Medoide solange mit Nicht-Medoiden ausgetauscht, bis ein gewisses Maß an „Kompaktheit“ für jeden Cluster und das gesamte Clustering erreicht ist [VHG03].

CLARA (Clustering Large Applications) ist eine Implementierung von PAM, die viele Stichproben aus den Ausgangsdaten zieht, PAM auf die gezogenen Objekte ausführt und das beste Clustering aus diesen Stichproben liefert [VHG03].

Der Algorithmus **CLARANS** (Clustering Large Applications based on Randomized Search) unterscheidet sich von PAM durch die Einführung zweier weiterer Parameter. Anstatt den Algorithmus solange zu wiederholen bis ein zufrieden stellendes Maß an Kompaktheit erreicht ist, werden Parameter für die Zahl der Iterationen i und das Maximum der zu vergleichenden Nachbar-Objekte n miteinbezogen. Der Clusteringprozess verwendet einen Graphen aller Objekte, wobei jeder Knoten im Graph ein potentielles Ergebnis (Medoid) darstellt. Zufällig ausgewählte Knoten werden mit n Nachbarn verglichen. Liefert ein Nachbar ein kompakteres Ergebnis, so wird die Suche bei diesem Knoten fortgesetzt. Ansonsten stellt der aktuelle Knoten ein lokales Optimum dar. Dann wird erneut ein zufälliger Knoten ausgewählt [VHG03].

Die hier erwähnten Algorithmen sind alle iterativ partitionierende Clusteringverfahren. Es gibt noch weitere Methoden, wie z.B. das dichtebasierte partitionierende Clustering, das hierarchische Clustering oder das graphenbasierte Clustering. Details zu diesen Methoden können u.a. in [ZRL96], [KHK99] und [EKSX96] gefunden werden.

2.2.2 Hierarchische Clusteringverfahren

Hierarchische Clusteringverfahren bilden Clusterstrukturen, in denen die Cluster ineinander liegen (siehe Abbildung 4 ganz links).
In Anbetracht der Methode, welche die Clusterhierachie erzeugt, kann zwischen zwei Arten von Verfahren unterschieden werden:

- **Agglomerative Algorithmen (bottom-up)**
  Zu Beginn bildet jedes Objekt der Ausgangsdaten einen Cluster. In jeder Iteration werden die beiden sich am nächsten liegenden Cluster zusammengelegt

- **Divisive Algorithmen (top-down)**
  Bei dieser Strategie gehören alle Ausgangsdaten zu Beginn genau einem Cluster an. In jeder Iteration wird ein Cluster in zwei aufgesplittet.

Die folgenden Algorithmen sind eine Auswahl an repräsentativen hierarchischen Clusteringverfahren.

Der Algorithmus **BIRCH** nutzt als hierarchische Hilfsstruktur einen sog. CF-Baum, um die Ausgangsdaten dynamisch zu partitionieren. Der CF-Baum speichert die Cluster-Attribute und basiert auf einen Parameter für die Verzweigungen des Baumes B und einen Parameter als Schwellenwert T für den maximalen Radius der Instanzen im Blattknoten. Die Idee von BIRCH liegt darin, nur einmal über die Ausgangsdaten iterieren zu müssen, dabei inkrementell die Cluster aufzubauen und nur die

Parameter der Cluster abzuspeichern (nicht die Daten selbst). Darüber hinaus ist BIRCH der erste Clusteringalgorithmus, der Ausreißer effektiv bewältigen kann.
Weitere Details zu BIRCH sind dem Kapitel 4.2 zu entnehmen.

CURE versucht jeden Cluster durch eine bestimmte Anzahl von Punkten, die aus einer gut verteilten Menge von Datensätzen stammen, zu repräsentieren. Dies geschieht, indem die Ausgangsdaten solange um den Centroiden verkleinert werden, bis ein bestimmter Bruchteil an Punkten erreicht ist.

Für das Clustern von Boolean-Werten und kategorischen Daten bietet **ROCK** eine robuste Vorgehensweise. ROCK misst die Ähnlichkeiten von Nachbarpunkten und deren Verbindungen zu weiteren Punkten.

Es existiert noch eine Reihe weiterer Clusteringverfahren, z.B. graphenbasiertes Clustering, GRID-Clustering, Subspace-Clustering und Fuzzy-Clustering.
Genaueres zu diesen Verfahren und zu weiteren Data-Mining-Methoden kann unter Ester und Sander [ES00], Han und Kamber [HK00], Freitas [FR02] gefunden werden.

2.2.3 Vergleich der Hauptcharakteristika ausgewählter Clusteringmethoden

Nachdem die wichtigsten Vertreter von partitionierenden und hierarchischen Clusteringverfahren vorgestellt wurden, sollen nun anhand einer Tabelle die Hauptcharakteristika dieser Methoden übersichtlich gegenübergestellt werden, um eventuelle Vor- und Nachteile besser nachvollziehen zu können (vgl. Tabelle 1).

Kategorie	Partitionierend				
Name	Skalenniveau	Komplexität	Ausreißer	Input-Parameter	Ergebnis
K-Means	Numerisch	O(n)	Nein	Anzahl der Cluster	Clusterzentren
PAM	Numerisch	$O(k(n-k)^2)$	Nein	Anzahl der Cluster	Medoide der Cluster
CLARA	Numerisch	$O(k(40+k)^2+k(n-k)$	Nein	Anzahl der Cluster	Medoide der Cluster
CLARANS	Numerisch	$O(kn^2)$	Nein	Anzahl der Cluster, Anzahl der Nachbarn	Medoide der Cluster
Kategorie	Hierarchisch				
BIRCH	Numerisch	O(n)	Ja	Radius der Cluster, Verzweigungsfaktor	CF-Baum
CURE	Numerisch	$O(n^2\log(n))$	Ja	Anzahl der Cluster	Zuweisung der Daten zu Clustern
ROCK	Kategorisch	$O(n^2+nm_m m_a+n^2\log(n))$ mit m_m als die maximale Anzahl und m_a als die durchschnittliche Anzahl von Nachbarn eines Punktes	Ja	Anzahl der Cluster	Zuweisung der Daten zu Clustern

Tabelle 1: Partitionierende und hierarchische Clusteringverfahren im Vergleich (vgl. [VHG03])

2.3 Klassifikation

Im Gegensatz zu Clusterbildungs- (Clustering-) Verfahren, die versuchen ohne Vorgaben Objekten Klassen zuzuweisen, sind die Klassen bei der Verwendung von Klassifikationsverfahren a priori bekannt. Objekte (Daten) werden zu vorgegebenen Klassen zugeordnet. Dazu ist eine Menge von sog. Trainingsobjekten (Trainingsdaten), die bereits klassifiziert sind, gegeben, anhand derer eine Funktion (Klassifikator) gelernt werden soll, der andere/neue und unklassifizierte Objekte aufgrund ihrer Attributsausprägungen einer Klasse zuordnen soll. Dabei ist es die primäre Aufgabe der Klassifikation [ES00]:

- Objekte aufgrund der Ausprägungen der Attributwerte einer vorgegebenen Klasse zuzuordnen.

- Generieren von Klassifikationswissen (explizites Wissen) über die Klassen in Form eines Klassifikators (z.B. Entscheidungsbaum) → „Knowledge Discovery“

Abbildung 5 veranschaulicht die Struktur eines Klassifikators mit seinen Eingangs- und Ausgangsdaten.

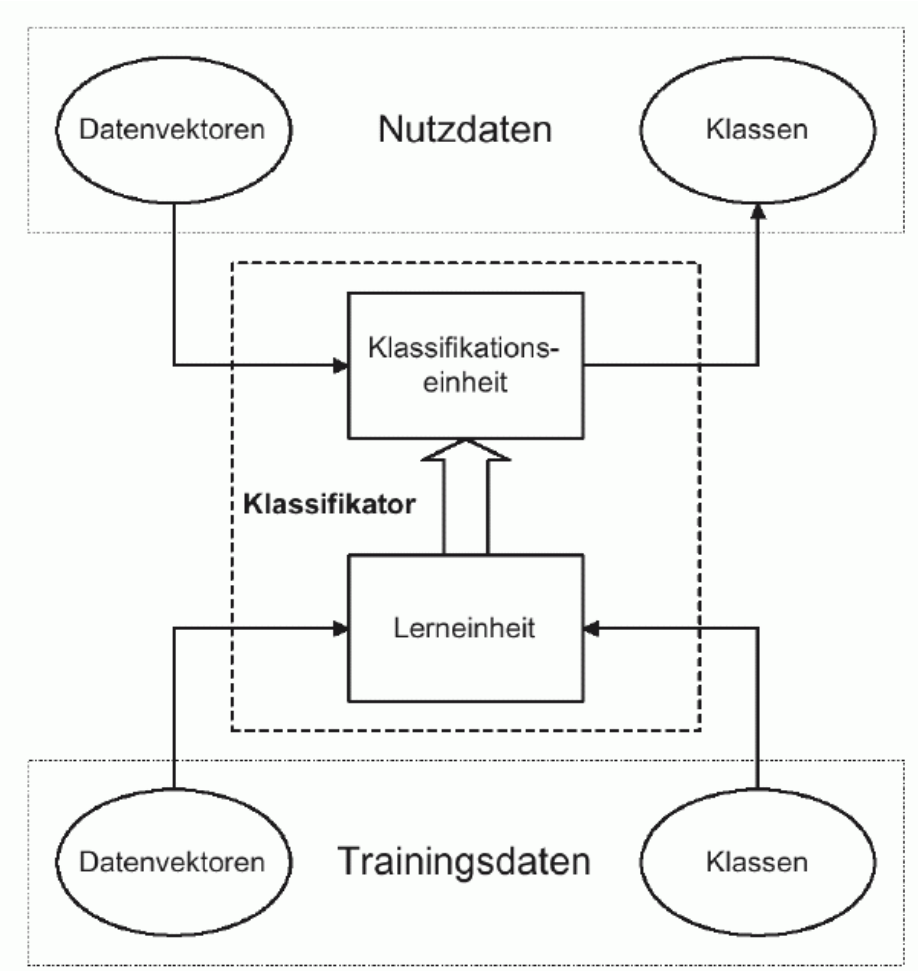


Abbildung 5: Struktur eines Klassifikators

Für die Klassifikation gehen wir von einer Menge O von Objekten $o = (o_1,\ldots,o_d)$ aus, von denen wir die Werte von d für die Klassifikation relevanten Attributen A_i, $1 \leq i \leq d$, kennen und die Klasse c_i, $c_i \in C = \{c_1,\ldots,c_k\}$, des jeweiligen Objekts (Tupel).
Attribute können kategorischen (ohne Ordnung des Wertebereichs, im Allgemeinen wenige verschiedene Werte) oder numerischen (mit totaler Ordnung des Wertebereiches, im Allgemeinen viele verschiedene Werte) Ursprungs sein. Die Klasse eines jeden Objekts wird im Normalfall durch ein zusätzliches Attribut

(Klassenattribut oder Zielattribut) angegeben. Tabelle 2 enthält eine Menge von Trainingsdaten zur Generierung eines Klassifikators, der in Abhängigkeit des Alters und des Familienstandes Personen in die Kategorien „hat Kinder“ bzw. „hat keine Kinder“ einordnen soll (Kinder „ja/nein“).

ID	Alter	Familienstand	Kinder
1	23	geschieden	nein
2	17	ledig	nein
3	43	ledig	nein
4	68	geschieden	ja
5	32	verheiratet	ja

Tabelle 2: Trainingstabelle „Kinder (ja/nein)“

Anhand der gegebenen Objekte soll ein Klassifikator als Funktion K der Form K: D → C erzeugt werden, wobei D $\supseteq$ O ist und die Klassenzugehörigkeit nur in O, nicht jedoch in D \ O bekannt ist [ES00].
Die Trainingsdaten aus Tabelle 2 sind beispielsweise durch folgende Regeln als Klassifikator gegeben:

If Alter > 50 then Kinder = ja;
If Alter ≤ 50 and Familienstand = verheiratet then Kinder = ja;
If Alter ≤ 50 and Familienstand ≠ verheiratet then Kinder = nein.

Die Klassifikation ist also ein Prozess, der aus zwei Phasen besteht. Die erste Phase besteht darin, ein Modell, den Klassifikator, aus vorklassifizierten Daten, den Trainingsdaten, zu erstellen. In der zweiten Phase verwendet man dieses Modell, um zu Klassifizieren. Als erstes muss aber, um den Klassifikator effizient nutzen zu können, die Klassifikationsgenauigkeit des Modells abgeschätzt werden. Kapitel 3.1 beschreibt dazu einfache Techniken unter Verwendung von vorklassifizierten Testdaten. Erst wenn das Modell eine ausreichend große Genauigkeit (geringe Fehlerwahrscheinlichkeit) besitzt, kann es eingesetzt werden, um zukünftige bzw. unklassifizierte Daten in Klassen zu gruppieren.

Die nun folgenden Kapitel fassen die gängigsten Klassifikationsverfahren zusammen. Informationen im Detail zu weiteren Verfahren wie z.B. Support Vector Machines oder Neuronale Netze sind unter [BU02] oder [HK00] zu finden.

2.3.1 Bayes-Klassifikation

Bayes-Klassifikatoren beruhen auf der Bestimmung der bedingten Wahrscheinlichkeiten der Attributwerte der verschiedenen Klassen [ES00]. Bayes-Klassifikatoren basieren auf dem „Bayes-Theorem“, welches anhand des nun folgenden „optimalen Bayes-Klassifikator“ genauer erläutert wird.

Der optimale Bayes-Klassifikator

Um den optimalen Bayes-Klassifikator einzuführen gehen wir von folgendem Beispiel aus:
X ist ein zufälliger Datensatz, dessen Klassenzugehörigkeit unbekannt ist. Die Hypothese H besagt, dass X zur Klasse C zuzuordnen ist. Um der Klassifikation gerecht zu werden, versucht man die bedingte Wahrscheinlichkeit P(H|X) zu ermitteln, die Wahrscheinlichkeit, dass die Hypothese unter Betrachtung von X verifiziert werden kann. Die bedingte Wahrscheinlichkeit P(H|X) wird „a posteriori Wahrscheinlichkeit“ von H genannt.
Das Objekt X wurde beispielsweise aus einer Datenmenge D von Fahrzeugen gezogen, die durch ihre Achsenanzahl beschrieben werden. Wenn nun Fahrzeug X vier Achsen besitzt, dann spiegelt P(H|X) die Sicherheit wider, dass X ein LKW ist. Im Gegensatz dazu ist P(H) die „a priori Wahrscheinlichkeit“ von H, die Wahrscheinlichkeit, dass ein Objekt der Klasse C zuzuordnen ist. Im aktuellen Beispiel drück P(H) die Wahrscheinlichkeit aus, dass die Datensätze aus D LKWs sind.
Die Wahrscheinlichkeit P(X) zeigt den Anteil der Fahrzeuge aus D an, die vier Achsen besitzen, während P(X|H) aussagt, wie groß der Anteil der 4-achsigen LKWs ist.
Mithilfe des „Bayes Theorems“ ist man in der Lage die „a posteriori Wahrscheinlichkeit“ P(H|X) zu berechnen [HK01]:

$$P(H \mid X) = \frac{P(X \mid H) \cdot P(H)}{P(X)} \tag{2.1}$$

Dies ist jedoch ein Spezialfall des optimalen Bayes-Klassifikators, bei dem die Annahme getroffen werden muss, dass immer genau eine Hypothese gültig ist für den Fall, dass mehrere Hypothesen aufgestellt wurden.

Der naive Bayes-Klassifikator

Eine vereinfachte Form des Bayes-Klassifikators stellt der sog. „naive Bayes-Klassifikator“ dar. Der naive Bayes-Klassifikator geht davon aus, dass der Effekt einer Attributausprägung auf die gegebene Klasse unabhängig von den Ausprägungen der anderen Attribute ist. Diese Annahme - auch „class conditional independence“ genannt – wurde aus Gründen der Vereinfachung der Berechnungen getroffen und führte schlussendlich dazu, dass diese Vorgehensweise als „naiv“ bezeichnet wurde [HK01].
Nun soll gezeigt werden, wie das Bayes´sche Theorem anhand des naiven Bayes-Klassifikators funktioniert:

1. Jedes Tupel wird durch einen n-dimensionalen Feature-Vektor repräsentiert, X = (x_1, x_2, …, x_n), wobei jede Ausprägung zu einem von n Attributen gehört, A_1, A_2, …, A_n.

2. Es existieren m Klassen, C_1, C_2, …, C_m. Für einen unbekannten zufälligen Datensatz X ermittelt der naive Bayes-Klassifikator jene Klasse, deren a posteriori Wahrscheinlichkeit unter der Bedingung, dass X gilt, am höchsten ist. Daraus folgt, dass ein unbekannter Datensatz X genau dann einer Klasse C_i zugeordnet wird, wenn

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

   Die Klasse C_i, die $P(C_i|X)$ maximiert wird „maximale a posteriori Hypothese“ genannt und durch das Bayes´sche Theorem (Formel 2.1) gilt:

   $$P(C_i \mid X) = \frac{P(X \mid C_i) \cdot P(C_i)}{P(X)}. \qquad (2.2)$$

3. P(X) ist konstant für alle Klassen. Sind die a priori Wahrscheinlichkeiten der Klassen unbekannt, dann wird im Normalfall davon ausgegangen, dass $P(C_1) = P(C_2) = \ldots = P(C_m)$ ist und daher $P(X|C_i)$ maximiert werden muss. Ansonsten wird $P(X|C_i)$. $P(C_i)$ maximiert und die a priori Wahrscheinlichkeiten der Klassen werden mit $P(C_i) = s_i / s$ bestimmt, wobei si die Anzahl der Datensätze von C_i und s die Gesamtanzahl der Trainingsdatensätze bezeichnet.

4. Die Berechnung von $P(X|C_i)$ wäre sehr Laufzeitintensiv, weshalb man die Annahme der „class conditional independence“ trifft. Im Falle einer Unabhängigkeit der Attributausprägungen von einander gilt:

$$P(X \mid C_i) = \prod_{k=1}^{b} P(x_k \mid C_i). \qquad (2.3)$$

   Sind die Attribute kategorisch skaliert, dann können die bedingten Wahrscheinlichkeiten $P(x_k|C_i)$ aus dem Quotienten der Anzahl der Trainingsdatensätze der Klasse C_i, die den Wert x_k für A_k besitzen und der Gesamtanzahl der Trainingsdatensätze berechnet werden.
   Sind die Attributausprägungen stetig numerisch so errechnet sich die bedingte Wahrscheinlichkeit $P(x_k|C_i)$ durch die Gauss´sche Glockenkurve (Normalverteilung) wie folgt:

$$P(x_k \mid C_i) = \frac{1}{\sqrt{2\pi\sigma_{C_i}}} e^{-\frac{(x_k-\mu_{C_i})^2}{2\sigma_{C_i}^2}}, \qquad (2.4)$$

   wobei μ_{Ci} und δ_{Ci} Erwartungswert und Standardabweichung der Ausprägungen von A_k der Trainingsdatensätze der Klasse C_i sind.

5. Um nun einen unbekannten Datensatz X zu klassifizieren, wird $P(X|C_i)$. $P(C_i)$ für jede Klasse C_i berechnet und X der Klasse C_i genau dann zugewiesen, wenn

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

Das heißt, dass X der Klasse C_i zugewiesen wird für die $P(X|C_i)$. $P(C_i)$ ein Maximum ist [ES00], [HK01].

2.3.2 Nearest-Neighbour-Klassifikation

Untypisch für Klassifikationsverfahren verzichtet die Nearest-Neighbour-Klassifikation (Nächste-Nachbarn-Klassifikation) auf das Finden von explizitem Wissen (Bayes-Klassifikator, Entscheidungsregeln, Entscheidungsbaum, …) und wird stattdessen direkt auf die Trainingsdaten angewandt. Aus diesem Grund werden Nächste-Nachbarn-Klassifikatoren auch „lazy lerners“ genannt, da sie keinen Klassifikator erzeugen ehe ein neues, unklassifiziertes Objekt klassifiziert werden muss [HK00].
Die verwendeten Trainingsobjekte werden durch einen n-dimensionalen Feature-Vektor repräsentiert, $X = (x_1, x_2, \ldots, x_n)$, wobei jede Ausprägung zu einem von n Attributen gehört, $A_1, A_2, \ldots, A_n$. Für jede Klasse Ci wird der Mittelwertvektor der zugehörigen Feature-Vektoren bestimmt. Die zu klassifizierenden Objekte werden der Klasse C_i des nächstgelegenen Mittelwertvektors μ_i zugeordnet.

Bei der Anwendung des Nächste-Nachbarn-Klassifikators sind drei einfache Regeln entscheidend:

- Es soll mehr als ein Trainingsobjekt pro Klasse verwendet werden.
- Als Entscheidungsmenge soll nicht nur ein nächster Nachbar dienen, sondern die k > 1 nächsten Nachbarn.
- Die Klassen der k nächsten Nachbarn sollten gewichtet werden, z.B. nach Distanz der Nachbarn vom zu klassifizierenden Objekt.

Abbildung 6 zeigt ein Beispiel für die Klassifikation eines Objekts anhand dieser Regeln.

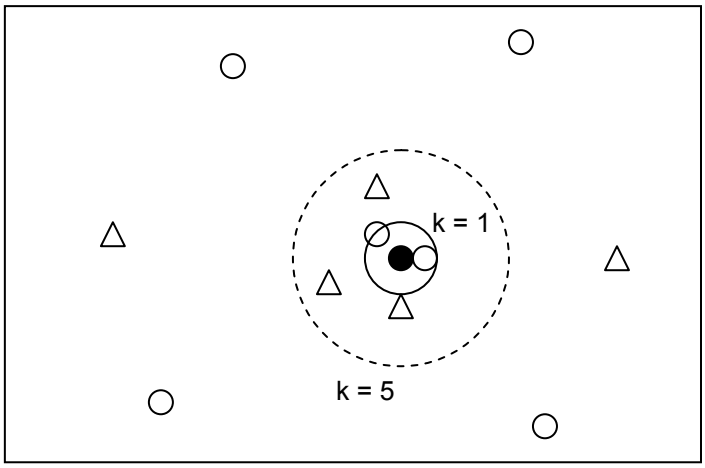


Abbildung 6: Nächste-Nachbarn-Klassifikation

Sind alle Objekte der Entscheidungsmenge gleich gewichtet, dann wird die Klasse des zu klassifizierenden Objekts o nach der Anzahl der Objekte der Entscheidungsmenge bestimmt: k = 1 → o = „○“, k = 5 → o = „Δ“. Werden die Objekte der Entscheidungsmenge nach ihrer Distanz von Objekt o gewichtet, so wird o in beiden Fällen von k zu „+“ klassifiziert. Zur Bestimmung einer solchen Distanz wird die Euklid´sche Distanz verwendet, die sich folgendermaßen berechnet:

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.5}$$

Der Nearest-Neighbour-Klassifikator erzeugt, wie bereits erwähnt, kein explizites Wissen, er liefert jedoch eine implizite Beschreibung der Klassen der Trainingsdaten in Form von Entscheidungsflächen, welche die Klassen „räumlich“ trennen (Abbildung 7) [ES00].

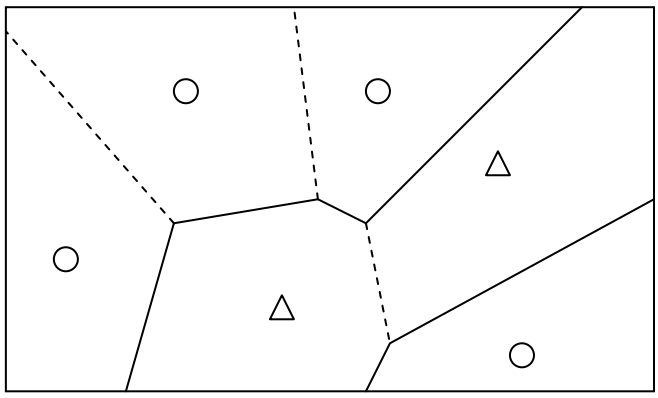


Abbildung 7: Entscheidungsflächen des Nächste-Nachbarn-Klassifikators

Bei der Verwendung von k = 1 Nachbarn wird das zu klassifizierende Objekt einfach der Klasse des nächst gelegenen Nachbarn zugeordnet. Diese Vorgehensweise ist vergleichbar mit der Zuordnung eines unklassifizierten Objekts zum nächsten Centroiden/Medoiden beim K-Means-Clustering. Erst bei Anwendung von k > 1 Nachbarn besteht ein Unterschied zwischen den beiden Data-Mining-Verfahren. Während beim Clustering das unklassifizierte Objekt immer dem nächsten Clustermittelpunkt zugeordnet wird, erfolgt die Klassifizierung bei der Nearest-Neighbour-Klassifikation anhand des durchschnittlichen Abstandes zu allen umliegenden Objekten.
Des Weiteren können Nächste-Nachbarn-Klassifikatoren auch zur Prognose (prediction) von Klassen herangezogen werden. Dies geschieht, indem einem unklassifizierten Objekt der Durchschnittswert der echten Klassen, der k nächsten Nachbarn des Objekts zugewiesen wird.

2.3.3 Entscheidungsbaum-Klassifikation

Das Entscheidungsbaum-Klassifikation (decision tree classification) liefert explizites Wissen zur Klassifikation in Form eines Entscheidungsbaumes. Durch die Erstellung eines Entscheidungsbaumes als Klassifikator erhält man als Ergebnis eine Schablone, die es ermöglicht neue bzw. zusätzliche Datensätze in das vorhandene Regelsystem, das aus einer ausgewählten Menge von Datensätzen (Trainingsmenge) produziert wurde, einzuordnen.

Definition

Ein Entscheidungsbaum ist ein Baum mit folgenden Eigenschaften:

- Ein innerer Knoten repräsentiert ein Attribut.
- Ein Blatt repräsentiert einer der Klassen.
- Eine Kante repräsentiert einen Test auf dem Attribut des Vaterknotens [ES00].

In Anlehnung an das Beispiel aus Kapitel 2.3 lässt sich aus den dort aufgestellten Entscheidungsregeln folgender Entscheidungsbaum erzeugen (vgl. Abbildung 8):

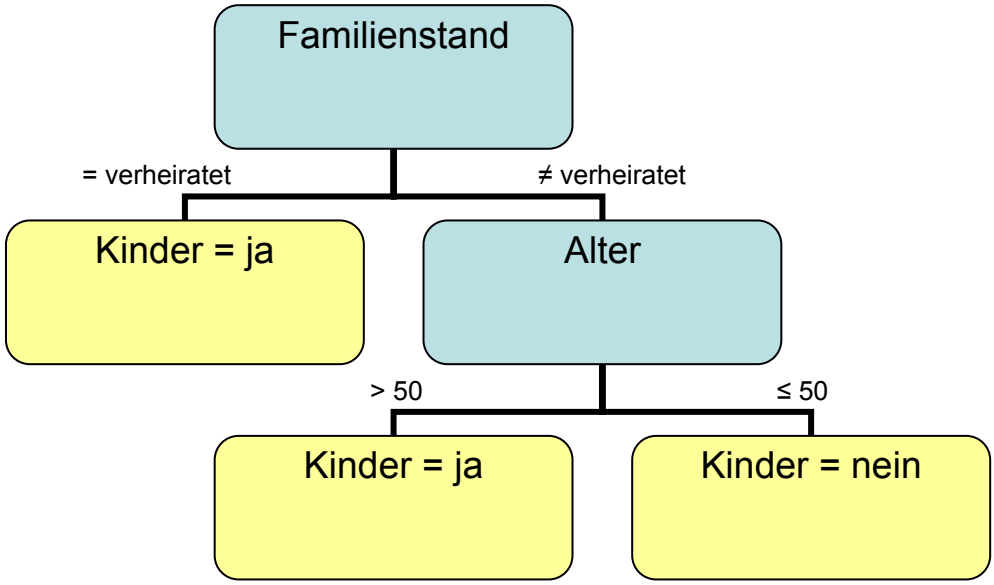


Abbildung 1: Entscheidungsbaum

Ein abstrakter Algorithmus zur Erzeugung eines Entscheidungsbaumes aus gegebenen Trainingsdaten könnte nach [ES00] so aussehen (vgl. Abbildung 9):

```
EntscheidungsbaumKonstruktion (Trainingsmenge T, Float min_confidence)
if mindestens min_confidence der Objekte aus T in Klasse c then
        return;
else
        for each Attribut A do
                for each möglicher Split von A do
                        bewerte die Qualität der Partitionierungen (T1,T2, …, Tm), die durch die
                        Splits entstehen würden;
        führe den besten aller dieser Splits durch;
        EntscheidungsbaumKonstruktion (T1, min_confidence);
        …
        EntscheidungsbaumKonstruktion (Tm, min_confidence);
```

Abbildung 2: Entscheidungsbaum-Algorithmus in Pseudocode

Der obige Algorithmus erzeugt solange rekursiv neue Sohnknoten aufgrund des besten Splits einer Dimension, bis das Kriterium „min_confidence“ am aktuellen Knoten erfüllt ist. Min_confidence drückt den Prozentsatz aus, den die Mehrheitsklasse an einem Knoten einnimmt. Die Mehrheitsklasse ist jene Klasse, der die meisten Trainingsdatensätze am Knoten zugeordnet werden.

Entscheidungsbaum-Klassifikatoren führen in der Regel binäre Splits durch (m=2), manche Algorithmen bilden aber auch ternäre oder n-äre Splits. Sie verwenden den sog. „Greedy-Algorithmus“, bei dem jeweils nur das nächste Split-Attribut ausgewählt wird. Nach der Aufbauphase des Entscheidungsbaums (Growth Phase) wird ein Baum meist beschnitten (Pruning Phase), um dem Phänomen des Overfitting (siehe Kapitel 3.3) entgegen zu wirken [ES00].

Splitstrategien

Die Splitstrategie ist das Herz des Entscheidungsbaum-Klassifikators. Sie wird dazu verwendet, ein Test-Attribut für jeden Knoten des Baumes zu finden. Vor dem Split müssen der Typ des Splits (binär, ternär, …, n-är) und Kriterien für die Qualität eines Splits festgelegt werden.
Entscheidungsbaum-Klassifikatoren eignen sich sowohl für kategorische als auch für numerische Daten. Bei einem Split über ein kategorisches Attribut wird entweder jede Attributsausprägung als Splitwert behandelt (n-ärer Split) oder es werden einzelne Ausprägungen zu Mengen zusammengefasst (binärer Split). Handelt es sich um einen Split über ein numerisches Attribut A, welches innerhalb ihres Wertebereichs eine totale Ordnung besitzt, so wird dieser Split im Allgemeinen von der Form (A ≤ Splitwert) || (A > Splitwert) sein. Ein mehrwertiger Split lässt sich z.B. mit der Struktur ($\text{Splitwert}_1 \leq A < \text{Splitwert}_2$ || ($\text{Splitwert}_2 \leq A \leq \text{Splitwert}_3$) || ($A > \text{Splitwert}_3$) darstellen.

Um nun feststellen zu können, welches Test-Attribut den „besten“ Split liefert, benötigt man ein Qualitätskriterium, das Auskunft über die Reinheit der Partitionen in Bezug auf die Klassenzugehörigkeit geben kann. Die gebräuchlichsten Kriterien zur Bestimmung der Qualität eines Splits sind der sog. „Informationsgewinn“ und der „Gini-Index“ [ES00]. Im Folgenden wird der Informationsgewinn als Kriterium eingeführt, da er auch später in dieser Arbeit verwendet wird:

Die **Entropie** für eine Menge T von Trainingsobjekten ist definiert als

$$entropie(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i, \qquad entropie(T) \in [0,1]. \tag{2.6}$$

Die Entropie ist ein Maß für die Unreinheit einer Verteilung, d.h. z.B. für entropie(T) = 0, dass eine Verteilung mit einem Splitwert S absolut rein ist und alle Objekte verschiedener Klassenzugehörigkeiten von einander getrennt wurden, während bei entropie(T) = 1 alle durch S erzeugten Partitionen die gleiche Menge an Objekten mit unterschiedlicher Klassenzugehörigkeit beinhalten.

Der Informationsgewinn eines Attributs A in Bezug auf die Menge T ist nach [ES00] definiert als

$$informationsgewinn(T,A) = entropie(T) - \sum \frac{|T_i|}{|T|} \cdot entropie(T_i), \tag{2.7}$$

wobei das Attribut A jenes Attribut ist, dessen Attributausprägungen mittels eines Splitkriteriums in die vollständigen, disjunkten Partitionen T_1, …, T_i geteilt wurden.

Die Anwendung des Informationsgewinns auf eine Trainingsmenge T soll im nun folgenden Beispiel illustriert werden (vgl. Tabelle 3), wobei als Trainingmenge wiederum die Datensätze der Tabelle 2 aus Kapitel 2.3 verwendet wird.

ID	Alter	Familienstand	Kinder
1	23	geschieden	nein
2	17	ledig	nein
3	43	ledig	nein
4	68	geschieden	ja
5	32	verheiratet	ja

Tabelle 3: Trainingstabelle für den Entscheidungsbaum-Klassifikator

Zunächst muss die Gesamtentropie der Trainingsmenge (entropie(T)) berechnet werden. T enthält zwei Tupel mit „Kinder = ja“ und drei Tupel mit „Kinder = nein“. entropie(T) ergibt sich somit aus:

$$entropie(T) = \frac{2}{5} \cdot \log_2(\frac{2}{5}) + \frac{3}{5} \cdot \log_2(\frac{3}{5}) = 0{,}97095$$

Im nächsten Schritt wird nun für jedes potentielle Split-Attribut der Informationsgewinn für einen oder mehrere Split-Punkt-Kandidaten berechnet (vgl. Tabellen 4 und 5).

Alter:

Alter	Kinder = nein	Kinder = ja	Entropie
> 50	0	1	0 = Partitionen rein
≤ 50	3	1	$\frac{3}{4}\cdot\log_2(\frac{3}{4})+\frac{1}{4}\cdot\log_2(\frac{1}{4})=0{,}81128$
gesamt	3	2	0,81128

Tabelle 4: Entropien des Attributs „Alter“

$$informationsgewinn(T, Alter) = entropie(T) - \frac{1}{5}\cdot 0 - \frac{4}{5}\cdot 0{,}81128 = 0{,}32193$$

Familienstand:

Familienstand	Kinder = nein	Kinder = ja	Entropie
geschieden	1	1	1 = Partitionen absolut verteilt
ledig	2	0	0 = Partitionen rein
verheiratet	0	1	0 = Partitionen rein
gesamt	3	2	1

Tabelle 5: Entropien des Attributs „Familienstand“

$$informationsgewinn(T, Autotyp) = entropie(T) - \frac{2}{5}\cdot 1 - \frac{2}{5}\cdot 0 - \frac{1}{5}\cdot 0 = 0{,}570951$$

Das Attribut „Familienstand“ liefert den höchsten Informationsgewinn und wird deshalb als erstes Split-Attribut gewählt. Abbildung 10 zeigt den aus diesen Berechnungen entstehenden mehrwertigen (ternären) Entscheidungsbaum.

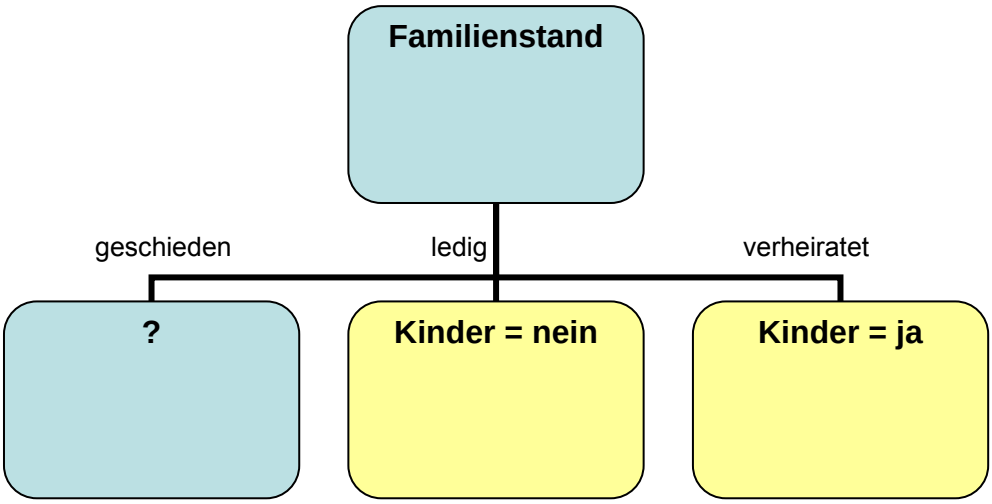


Abbildung 10: Mehrwertiger Entscheidungsbaum nach dem ersten Split

Die Partitionen „ledig“ und „verheiratet“ sind bereits rein und müssen kein weiteres Mal gesplittet werden. Die Partition „geschieden“ muss mithilfe des Informationsgewinns rekursiv solange weiter gesplittet werden bis sie rein ist bzw. ein oder mehrere andere Qualitätskriterien (min_support, min_confidence) erfüllt sind. (siehe Kapitel 3.3 zum Thema Pre-pruning).
Abbildung 11 präsentiert den fertigen Entscheidungsbaum mit reinen Partitionen.

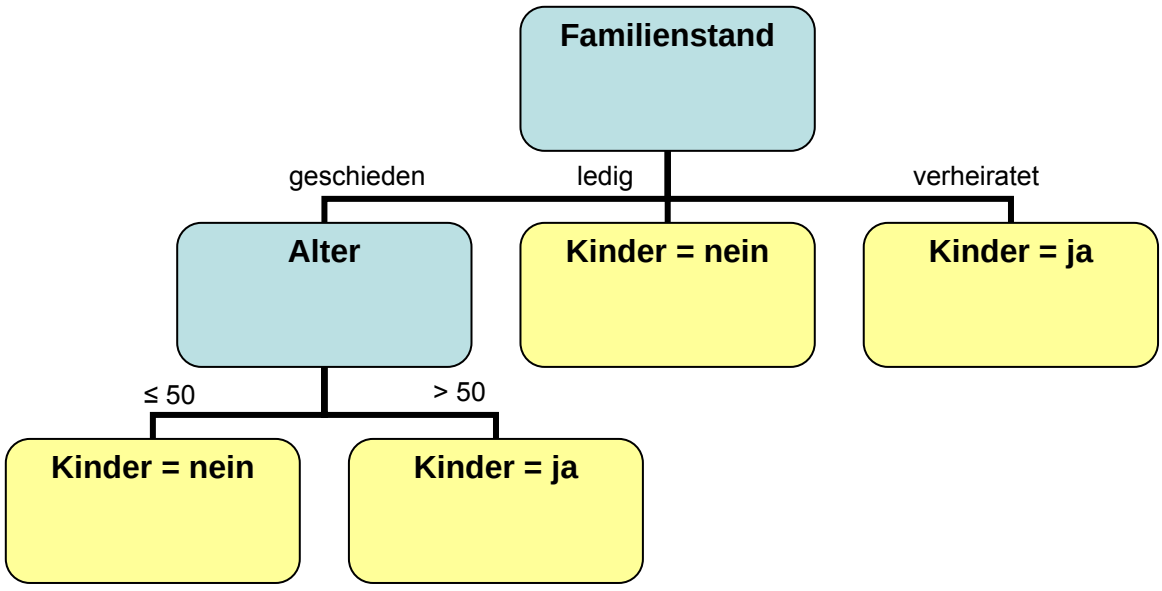


Abbildung 11: Mehrwertiger endgültiger Entscheidungsbaum

Ein so generierter Entscheidungsbaum muss, ehe er als gültig bezeichnet werden kann, auf seine Klassifikationsgenauigkeit überprüft werden [FR02]. Die bestehenden Möglichkeiten zur Bestimmung der Klassifikationsgenauigkeit bzw. weiterer

Qualitätsmerkmale von Klassifikatoren und Maßnahmen gegen Overfitting werden gesondert in Kapitel 3 erläutert. Die dort vorgestellten Qualitätskriterien sind Maße für die Qualität und Güte des gesamten Entscheidungsbaum-Klassifikators.

2.3.4 Klassifikationsregeln

Als Alternative zu Entscheidungsbäumen existiert das Verfahren der Generierung von Klassifikationsregeln [FR02], die bereits früher in diesem Dokument kurz umrissen wurden (siehe Kapitel 2.3).
Wie bei den Entscheidungsbäumen werden Attribute ausgewertet, die in den Bedingungen der Regeln auftreten. Diese Bedingungen sind häufig mit AND verknüpft, es sind jedoch andere boolesche Operatoren möglich (OR, XOR, NOT, …).
Allgemeine Beispiele für solche Regeln wären z.B.

IF A > 1 AND B ≥ 5 THEN c = 1,
IF C < 2 AND D > 2 THEN c = 2,

wobei A,B,C und D die Attribute einer Trainingsmenge darstellen und c die Klasse bezeichnet, der eine unklassifizierte Instanz (ein Testobjekt) zugewiesen werden würde.

Die Folgerung bei einer Erfüllung einer Regel ist die Klasse der Instanz.
Bei der Anwendung der Regeln können jedoch auch Probleme entstehen. Regeln können z.B. zueinander konfliktär stehen, was darauf zurückzuführen sein könnte, dass die Trainingsdaten Rauschen oder Fehler enthalten. Außerdem kann das Ergebnis von der Reihenfolge der angewendeten Regeln abhängig sein. Oftmals sind automatisch erzeugte Regeln für den Anwender auch nicht intuitiv verständlich.

Klassifikationsregeln können aus Entscheidungsbäumen erstellt werden oder umgekehrt. Die einfachere Variante ist dabei die **Umwandlung eines Baumes in eine Regelmenge**. Dabei gelten folgende Vorschriften:

- Erzeuge eine Regel pro Blatt.

    - Eine Regel entsteht durch die Verknüpfung der Bedingungen auf dem Weg von der Wurzel zum Blatt.
    - Der Schluss der Regel ist die Klasse des Blatts.
- Halte die Regeln möglichst einfach!
- Kürze die Regeln falls nötig, um Redundanzen auszuschließen!

Aus dem Entscheidungsbaum in Abbildung 11 lassen sich beispielhaft folgende Regeln erzeugen:

IF Familienstand = geschieden AND Alter > 50 THEN Kinder = ja;
IF Familienstand = ledig THEN Kinder = nein;

Etwas schwieriger gestaltet sich das **Verfahren vom Regelwerk zum Baum**. Eine Disjunktion lässt sich im Entscheidungsbaum nicht immer einfach ausdrücken. Als Beispiel wären mehrere Regeln mit verschiedenen Attributen zu nennen, welche die gleiche Ergebnisklasse zuordnen.

IF A > 1 AND B > 5 THEN c = 1,
IF C < 2 AND D > 2 THEN c = 1;

Die Regeln oben können nicht gemeinsam in einem Entscheidungsbaum repräsentiert werden, da ihnen das gemeinsame Attribut fehlt, dass als Wurzelknoten fungieren würde. Im Falle des „Kinder-Beispiels“ wäre es unmöglich, den vorhandenen Baum aus den folgenden Regeln zu rekonstruieren:

IF Familienstand = ledig THEN Kinder = nein;
IF Alter ≤ 50 THEN Kinder = nein;

Es gibt verschiedene Typen von Klassifikationsregeln [BL97], die nachfolgend kurz erläutert werden.

Bei den **klassischen Klassifikationsregeln** stellt jede Regel einen unabhängigen Wissensbaustein da und jeder Regelmenge können ohne Störung der Konsistenz neue Regeln hinzugefügt werden. Einzig und alleine die Abarbeitungsreihenfolge ist von Bedeutung, da es bei Nichtbeachtung der Regelfolge zum Konflikt zwischen

einzelnen Regelbausteinen kommen kann. Problematisch ist weiters, dass es keine festgeschriebene Vorgehensweise im Konfliktfall oder im Fall, dass keine Regel für eine Instanz passend ist, gibt. Mögliche Lösungsvorschläge wären, den häufigsten Schluss aus der Regelmenge anzugeben, oder die Instanz als „unklassifiziert“ zu markieren.

Bool´sche Klassifikationsregeln vereinfachen die Situation dahingehend, dass die triviale Annahme getroffen wird, eine Instanz gehöre zu ¬c („nein“), wenn sie nicht zu c („ja“) gehört. Daraus folgt, dass nur sog. „ja“-Regeln gelernt werden müssen, die kein Konfliktpotential besitzen und unabhängig von ihrer Reihenfolge angewandt werden können.

Die obigen Klassifikationsregeln bieten ausschließlich die Möglichkeit „proportionale“ Regeln zu erstellen, d.h. Attributausprägungen werden mit Konstanten verglichen. Dagegen bieten **Relationale Regeln** – wie der Name andeutet – eine relationale Lösung. Die Standardrelationen sind dabei <, > und = mithilfe derer Beziehungen zwischen Attributen modelliert werden können. Das Lernen von relationalen Regeln ist sehr aufwendig und fordert die Einführung von neuen Attributen, wie im folgenden Beispiel veranschaulicht.

IF Breite > Höhe THEN liegend,
IF Höhe > Breite THEN stehend.

Als neues Attribut wird hier z.B. „Breite<Höhe“ eingeführt, in dem die Klasse aller Test-Instanzen bezüglich der obigen Regeln eingetragen wird.

Alle Formen von Regeln können anhand von Kriterien bezüglich ihrer Qualität bewertet werden. Diese Kriterien sind beispielsweise die Abdeckung bzw. „coverage“ (= Anzahl der Instanzen, die korrekt vorhergesagt werden) oder die Genauigkeit bzw. „accuracy“ (= Verhältnis der Abdeckung zu allen Instanzen, auf die die Regel angesetzt wird). Normalerweise werden Vorgaben für coverage und accuracy gemacht, um nur solche Regeln zu erzeugen, die diese Vorgaben erfüllen.
Detailliertere Ausführungen zum Thema „Qualitätskriterien von Klassifikatoren“ werden im nächsten Kapitel (Kapitel 3) gegeben.

3 Bewertung von Klassifikatoren

Wie bereits in Kapitel 2.3 erwähnt müssen Klassifikatoren validiert werden, bevor sie auf neue unklassifizierte Objekte angewendet werden können. Die Trainingsmenge alleine ist kein guter Leistungsindikator [SC03]. Dies gilt im besonderen Falle dann, wenn nur eine kleine Menge an Daten zur Verfügung steht. Aber auch wenn ein genügend großes Trainingsset vorhanden ist, kann es sein, dass die Datenmenge wiederum nicht ausreicht, um Besonderheiten in den Daten zu identifizieren, die zwar regelmäßig aber nicht häufig auftreten. Fehlklassifikationen auf die Trainingsmenge, die sog. „beobachtbaren Fehler“, sind kein gutes Indiz für Leistung, da der Klassifikator die Charakteristika der Trainingsdaten erlernt hat und die Klassifikation eine „optimistische Schätzung“ darstellt. Daher ist es vorzuziehen, eine Leistungsbewertung an einer unabhängigen Testmenge vorzunehmen.

3.1 Bewertungsverfahren

Zur Bewertung von Lernverfahren stehen verschiedenste Herangehensweisen zur Verfügung. In der Laborstudie ist es schwierig bis unmöglich, Klassifikatoren so eingehend zu testen wie das beim Einsatz im Feld der Fall ist. Im Labor wird der vorhandene Datenbestand in Trainings- und Testdaten unterteilt, während in der realen Welt eine dritte Menge, neue und unklassifizierte Objekte, als Testmenge vorhanden sind, die erst bei längerem Einsatz des Klassifikators Aussagen über die wirkliche Leistungsfähigkeit treffen lassen [FR02].
In Abhängigkeit der für das Training vorgesehenen Daten, wird eine der folgenden Methoden angewandt.

Die sog. **Holdout-Methode** wird sehr oft bei großen Datenmengen verwendet. Die Bewertung erfolgt folgendermaßen [FR02]:

- Im ersten Schritt wird eine zufällig gezogene Partition der Ausgangsdaten in Lerndaten (Trainingsdaten) und Testdaten aufgeteilt. Eine bestimmte Datenmenge wird (in der Praxis meist 1/3) zurückgehalten, um die Konzeptbeschreibung an einer unabhängigen Testmenge zu erproben.

- Anschließend werden Fehlerschätzungen aufgrund der Testdaten vorgenommen.
- Unter der Voraussetzung, dass die Partition einer repräsentativen Stichprobe der Ausgangdaten entspricht, liefert dieses Verfahren eine pessimistische Schätzung.

Zur Verbesserung der Stichprobe wird in weiterer Folge ein **stratifiziertes Holdout-Verfahren** angewendet [FR02]. Bei der Stratifikation (Schichtenbildung) wird die Stichprobe auf ein richtiges Verhältnis der Klassen hin untersucht.
Eine weitere Möglichkeit zur Kompensation des Fehlers der Stichprobe ist die Verwendung eines **wiederholten Holdouts.** Dabei wird bei jeder Wiederholung des Trainings ein bestimmter Bruchteil der Zufallspartition herangezogen und anschließend die durchschnittliche Fehlerrate als Ergebnis ausgewiesen.

Bei der Holdout-Methode dient immer genau eine Partition als Messgrundlage der Klassifikationsgenauigkeit. Die sog. **Kreuzvalidierung** hingegen teilt die Ausgangsdaten zu Beginn in k in etwa gleich große, disjunkte und vollständige Partitionen [FR02]. Der Parameter k ist benutzerdefiniert, wobei sich ein Wert von k = 10 in der Praxis bewährt hat (stratifizierte zehnfache Kreuzvalidierung). Dann wird der Klassifikationsalgorithmus mit k-1 Partitionen als Trainingsmenge ausgeführt. Das Verfahren wird dann k-mal wiederholt, wobei jede Untermenge einmal als Testmenge eingesetzt wird.

Eine Spezialform der Kreuzvalidierung ist die sog. **„leave-one-out"-Methode** [FR02]. Bei diesem Verfahren wird der Parameter k mit der Anzahl der zur Verfügung stehenden Datenobjekte gleich gesetzt. Bei jedem Lauf des Klassifikationsalgorithmus dienen dann k-1 Datensätze als Trainingsmenge. Ein Datensatz wird ausgelassen – daher der Name „leave-one-out" – und als Testinstanz verwendet. Der Vorteil dabei liegt in der Tatsache, dass die Anzahl der zum Training verwendeten Instanzen pro Partition maximiert wird. Ein entscheidender Nachteil ist jedoch der immense Rechenaufwand.

Die Qualitäts- (Performance-) Bewertung bei Klassifikationsverfahren ist die Grundlage für die Klassifikator-Auswahl. Die Genauigkeitsabschätzung aus den

Lerndaten kann z.B. durch die Eintragung der vorhergesagten Klassen in eine wie in Tabelle 6 dargestellte Fehlklassifikationstabelle erfolgen.

		Vorausgesagte Klasse						
		2		.	.	.	g	Total
Korrekte Klasse	1	n_{11}	n_{12}				n_{1g}	n_1
	2	.	.					n_2
	.	.	.					.
	.	.	.					.
	.	.	.					.
	g	n_{g1}	.	.	.	.	n_{gg}	n_g / N

Tabelle 6: Fehlklassifikationstabelle

Im Falle dieser Arbeit stand eine ca. 6.000.000 Tupel große normalverteilte und vorklassifizierte Datenmenge zur Verfügung. Da durch die Testläufe nicht alleine die Qualität und Leistung des Klassifikators ermittelt werden sollte, sondern auch der Einfluss der verwendeten Hilfsinformationen auf das Ergebnis des Klassifikations-Algorithmus, wurde aus der Gesamtdatenmenge eine repräsentative Stichprobe von jeweils 0,1 % pro erstelltem Klassifikator (hier ein Decision Tree Classificator) nach der Holdout-Methode als Trainings- (1/3) und Testset (2/3) gezogen. Weitere Details zur Beschaffenheit der Trainings- und Testdaten sind dem Kapitel 8.1 zu entnehmen.

3.2 Qualitätskriterien von Klassifikatoren

Unter Qualitätskriterien werden genau solche Faktoren zusammengefasst, die Aussagen über die Qualität und Güte des generierten Klassifikators treffen. Da sich die Qualität eines Klassifikators aber nicht immer direkt, d.h. am Klassifikator selbst, messen lässt, ist es oftmals notwendig Rückschlüsse aus den klassifizierten Daten zu ziehen.

Im Anschluss erfolgt eine Aufzählung der Gütemaße von DT-Klassifikatoren, von denen jedoch nur Ausgewählte in dieser Arbeit zur Bestimmung der Qualität herangezogen werden:

Klassifikationsgenauigkeit

Das wichtigste Qualitätsmaß für DT-Klassifikatoren ist die Klassifikationsgenauigkeit. Sie trifft Aussagen darüber, welcher Anteil der klassifizierten Testdaten richtig eingeordnet wurde. Ein Klassifikator K mit einer Trainingsmenge TR und einer Testmenge TE (beide Mengen aus dem Objektraum O) erzeugt für ein Objekt o die zugehörige Klasse C(o). Die Klassifikationsgenauigkeit und die korrespondierenden Klassifikationsfehler berechnen sich folgendermaßen:

- **Klassifikationsgenauigkeit** (classification accuracy) von K auf TE:

  $$G_{TE}(K) = \frac{|\{o \in TE \mid K(o) = C(o)\}|}{|TE|}$$

  Die Klassifikationsgenauigkeit ist ein Maß für den Anteil der richtig klassifizierten Testdaten in Bezug auf eine unabhängige Testmenge.

- **Tatsächlicher Klassifikationsfehler** (true classification error) von K auf TE:

  $$F_{TE}(K) = \frac{|\{o \in TE \mid K(o) \neq C(o)\}|}{|TE|}$$

  Der tatsächliche Klassifikationsfehler ist ein Maß für den Anteil der falsch klassifizierten Testdaten in Bezug auf eine unabhängige Testmenge.

- **Beobachteter Klassifikationsfehler** (apparent classification error) von K auf TR:

  $$F_{TR}(K) = \frac{|\{o \in TR \mid K(o) \neq C(o)\}|}{|TR|}$$

  Der beobachtete Klassifikationsfehler ist ein Maß für die Fehlklassifikation der zur Konstruktion des Klassifikators verwendeten Trainingsdatensätze.

Wird der Baum soweit verfeinert, dass jedes Blatt nur noch Datensätze einer Klasse enthält, so ist der beobachtete Klassifikationsfehler immer 0. Ein bin ins Äußerste

verfeinerte Baum ist auf die Trainingsdaten hin optimiert und wird mit großer Wahrscheinlichkeit auf die Grundgesamtheit ein schlechtes Ergebnis liefern. Diesen Effekt bezeichnet man als Overfitting [ES00].

Unter Berücksichtigung der **Relevanz** der Testdatensätze können weitere Kennzahlen (Qualitätskriterien) mittels Bestimmung verschiedener Klassifizierungsparameter aus den klassifizierten Ergebnisdaten bestimmt werden. Um die Aussagekraft und den Wert dieser Kennzahlen besser begreifen zu können, wird im Vorfeld der Begriff der Relevanz definiert:

„Die Relevanz eines Dokuments für eine Anfrage ist eine Relation **r: D×Q -> R** , wobei **D = {d_1, ...,d_m}** die Menge der Dokumente, **Q** die Menge der Anfragen und **R** eine Menge von *Wahrheitswerten*, im Allgemeinen die Menge **{0,1}** , ist.“ [FE03]

Die Relevanz von Dokumenten wird allgemein durch die Befragung von Experten ermittelt. Handelt es sich beim Klassifikationsgegenstand, so wie im Falle dieser Arbeit, nicht um Dokumente, sondern um Datensätze aus numerischen und kategorischen Werten, so ist die Bestimmung der Relevanz auf diesem Weg nicht möglich. Deshalb wird als Annahme der Relevanz folgende Vereinbarung getroffen:

Ein vorklassifizierter Datensatz aus einer Testmenge T ist genau dann relevant, wenn die vom Testdatengenerator zugewiesene Klasse der durch das erste Verfahren des kombinierten Data-Mining-Prozesses ermittelten Klasse entspricht.

Die Tabelle 7 enthält die Klassifizierungsparameter, die zur Berechnung der Qualitätskriterien bzgl. der Relevanz benötigt werden.

	Eintrag gehört zu c	Eintrag gehört zu ¬c
Eintrag wurde zu c klassifiziert	a	b
Eintrag wurde zu ¬c klassifiziert	c	d

Tabelle 7: Klassifizierungsparameter

Aus diesen vier Werten lassen sich nun die Qualitätskriterien für die Genauigkeit eines Klassifikators K bezüglich der Relevanz der Testdaten berechnen:

- $precision(K) = \frac{a}{a+b}$

  Die **Präzision** (precision) einer Testmenge T ist der Quotient aus allen relevanten richtig klassifizierten Tupeln und allen richtig klassifizierten Tupeln.

- $recall(K) = \frac{a}{a+c}$

  Die **Vollständigkeit** (recall) einer Testmenge T ist der Quotient aus allen relevanten richtig klassifizierten Tupeln und allen relevanten Tupeln.

- $accuracy(K) = \frac{a+d}{a+b+c+d}$

  Die **Genauigkeit** (accuracy) einer Testmenge T ist daher das Verhältnis aller im Sinne der Relevanz richtig klassifizierten Datensätze zu allen Sätzen der Testmenge. Ein Datensatz ist im Sinne der Relevanz richtig klassifiziert, wenn er der Klasse c angehört (relevant) und der Klasse c zugeordnet wurde oder wenn er nicht der Klasse c angehört (nicht relevant) und der Klasse c nicht zugeordnet wurde.

Die Werte für precision und recall sind genau dann optimal, wenn alle relevanten Datensätze richtig klassifiziert wurden. Es ergibt sich für beide Kennzahlen ein Wert von 1. Die Maße für precision und recall sind in gewisser Weise gegenläufig. Wurden z.B. alle Tupel der Testmenge richtig klassifiziert, so wird der Wert für den recall 1 entsprechen, natürlich unter der Voraussetzung, dass mindestens ein relevanter Datensatz existiert. Die precision hingegen wird sehr niedrig sein, wenn nicht zufälligerweise alle Datensätze relevant sind. Sollte im umgekehrten Fall nur ein einziger relevanter Datensatz richtig klassifiziert worden sein, so ist die precision gleich 1, der recall wird aber mit Sicherheit schlecht sein, sollten noch weitere relevante Datensätze vorhanden sein.
In der Regel werden die Antwortmengen aber zwischen diesen beiden Extremen liegen. Dann ergibt sich im Allgemeinen bei einer Verkleinerung der Antwortmenge durch eine spezifischere Anfrage eine bessere Precision, aber ein schlechterer Recall; bei einer Vergrößerung der Antwortmenge durch eine allgemeinere Anfrage ergibt sich ein größerer Recall, aber eine kleinere Precision. Ähnliche Situationen können auftreten, wenn Precision- und Recall-Maße dazu verwendet werden, verschiedene Systeme zu vergleichen. Eindeutige Aussagen darüber, ob ein System besser ist als das andere, können nur gemacht werden, wenn für das eine System sowohl der Precision-Wert als auch der Recall-Wert besser ist als bei dem anderen System. Ist bei einem System z.B. die Precision besser, dafür aber der Recall schlechter, so eignen sich die Systeme zwar eventuell für unterschiedliche Aufgaben, es kann aber nicht allgemein gesagt werden, welches besser ist [FE03].

Kompaktheit des Modells

Ziel beim Generieren eines DT-Klassifikators ist es immer, diesen so kompakt als möglich zu halten, d.h. mit einem Minimum an Knoten ein Maximum an Qualität im Hinblick auf die Klassifikationsgenauigkeit zu erhalten. Zur Kompaktheit gehört außerdem die Höhe des Baumes. Es soll versucht werden, den Baum so „niedrig“ wie möglich zu halten, d.h. dass die Pfade eines Baumes nicht unnötig viele Knoten bzw. Verzweigungen enthalten. Je kompakter ein Modell ist, desto leichter kann es von einem Benutzer interpretiert werden.
Einerseits kann Kompaktheit während des Prozesses der Klassifikator-Konstruktion z.B. durch die Verwendung bestimmter Split-Strategien erreicht werden, andererseits

kann durch ein Fehlerreduktionspruning (siehe Kapitel 6.2.4) ex-post Einfluss auf die Kompaktheit des Baumes genommen werden.

Interpretierbarkeit des Modells

Die Interpretierbarkeit bezieht sich in erster Linie darauf, wie viel und welches (neue) Wissen ein Benutzer aus den Ergebnissen des durchgeführten Klassifikations-Verfahrens vermittelt bekommt. Außerdem wird Augenmerk auf die Einfachheit der Interpretation der Ergebnisse gelegt. Die Interpretierbarkeit hängt auch von der Kompaktheit des Modells ab. Unter Berücksichtigung der Klassifikationsgenauigkeit ist ein kompaktes Modell einem weniger kompakten vorzuziehen, da es sich leichter interpretieren lässt.

Effizienz

Mit Effizienz ist im Falle der Klassifikation in erster Linie die Laufzeit des Klassifikations-Algorithmus gemeint. Das Gütemaß der Effizienz teilt sich in die Effizienz der Konstruktion des Modells und die Effizienz der Anwendung des Modells. Im Falle des für die Verwertung von Hilfsinformationen optimierten Algorithmus, der darauf abzielt, eine Qualitätssteigerung in Form erhöhter Klassifikationsgenauigkeit zu erreichen, ist nicht unbedingt mit einer Effizienzsteigerung bei der Konstruktion des Modells zu rechnen. Es ist aber durchaus möglich, dass bestimmte Hilfsinformationen einen positiven Einfluss auf die Effizienz ausüben werden.
In der Anwendung sind Entscheidungsbaum-Klassifikatoren effizient und einfach zu handhaben. Die Testdaten durchlaufen von der Wurzel an top-down den Baum über einen eindeutigen Pfad und werden zur Klasse des erreichten Blattknotens zugeordnet.

Skalierbarkeit für große Datenmengen

Da die Datenmengen tendenziell ansteigen, ist es notwendig, sich Gedanken über die Reduktion des Gesamtaufwandes bei der Erzeugung des Klassifikators zu machen. Einflussfaktoren auf den Gesamtaufwand sind unter anderen:

- Die Größe der Trainingsmenge (alle Daten oder Stichprobe)
- Die Daten- und Indexstrukturen (alle Daten im Haupt- oder Sekundärspeicher)
- Die Evaluation der Splits
- Die Partitionierung der Trainingsmenge

Es existieren zwei Ansätze zur Skalierung von Entscheidungsbaum-Klassifikatoren [ES00]:

- **Sampling**
  Unter Sampling versteht man einerseits das Ziehen von Stichproben als Trainingsmenge aus der Datenbank, andererseits das Evaluieren einer Stichprobe aller potentiellen Splits. Durch Sampling verlieren Entscheidungsbäume an Qualität.

- **Unterstützung durch spezielle Daten- und Indexstrukturen**
  Bei diesem Ansatz wird die gesamte Datenbank als Trainingsmenge herangezogen. Sie wird im Sekundärspeicher gehalten und durch spezielle Daten- und Indexstrukturen unterstützt. Dieser Ansatz enthält keinen Unterschied zum speicherresistenten Ansatz bei der Konstruktion der Entscheidungsbäume und führt daher zu keinem Qualitätsverlust.

Robustheit

Unter Robustheit wird das Verhalten des Klassifikators gegenüber Rauschen und fehlenden Werten verstanden.
Im Fall des zu entwickelnden DT-Klassifikators wird dieses Gütemaß nicht berücksichtigt bzw. vorausgesetzt, da alle Trainings- und Testdaten aus dem Ergebnis des vorgelagerten Clustering-Algorithmus stammen und somit vorklassifiziert und vollständig sind.

3.3 Fehlerreduktionspruning bei Entscheidungsbäumen

Fehlerreduktionspruning wird mit der Grundidee durchgeführt, dass Entscheidungsbaum-Klassifikatoren im Normalfall für die Trainingsdaten optimiert sind. Es kann daher durchaus möglich sein, dass der Klassifikator auf die Grundgesamtheit der Daten schlechtere, d.h. ungenauere, Ergebnisse liefert [ES00]. Dieses Phänomen wird auch als „Overfitting“ bezeichnet. Overfitting tritt in immer dann auf, wenn die Trainingsdaten Rauschen bzw. Fehler enthalten oder wenn die Trainingsdaten keine repräsentative Stichprobe der Grundgesamtheit bilden.

Um Overfitting zu vermeiden, existieren zwei Ansätze des sog. „tree-prunings“ (Baumbeschneidung) [HK00]. Beim **Pre-pruning** wird versucht, die Konstruktionsphase so bald wie möglich zu stoppen. Genauer gesagt wird ein Knoten immer dann zu einem Blattknoten, wenn er Qualitätskriterien wie min_support oder min_confidence erfüllt bzw. nicht mehr erfüllt. Als Klasse am Blattknoten wird jene verwendet, die am häufigsten in den Ausprägungen am Knoten vorkommt.
Beim **Post-Pruning** werden Knoten bzw. Teilbäume des vollständig erzeugten Baumes entfernt. Das Fehlerreduktionspruning zielt dabei darauf ab, in mehreren Zyklen immer wieder den „schwächsten Link“ in einem DT-Klassifikator zu entfernen, um so die Qualität in Bezug auf die Klassifikationsgenauigkeit der Daten aus der Grundgesamtheit zu steigern.
Die Beschneidung des Entscheidungsbaumes erfolgt in folgenden Grundschritten:

1. Vorbereitung: Man teilt die Trainingsmenge in Trainings- und Testmenge auf und benutzt die Trainingsmenge um den Baum aufzubauen. Die Testmenge wird verwendet, um den Klassifikationsfehler zu messen.

2. Man ermittelt für alle Knoten des Baumes, inwieweit sich die Klassifizierungsqualität verbessern würde wenn man den jeweiligen Knoten und seinen Teilbaum weglassen würde.

3. Den Knoten (Teilbaum), bei dem sich die Klassifikation am meisten verbessern würde, lässt man wegfallen. Dies macht man so lange bis keine Verbesserung mehr möglich ist.

Pre-pruning und Post-pruning können auch als kombinierter Ansatz gemeinsam eingesetzt werden. Post-pruning ist in der Regel sehr ressourcenintensiv, führt aber im Großteil der Fälle zu einem zuverlässigeren Baum.
Im Fall des im Zuge dieser Arbeit implementierten Entscheidungsbaum-Klassifikators wurde aus Gründen der Performanz und der Vergleichsbasis der Ergebnisse gänzlich auf Post-pruning verzichtet und alle Pruningaktivitäten im Sinne des Pre-prunings auf die Phase des rekursiven Aufbaus des Baumes reduziert.

Weitere Aktivitäten zur Vermeidung von Overfitting sind die Wahl einer geeignet großen Trainingsmenge und das Entfernen von fehlerhaften/inkonsistenten Trainingsdaten. Die verwendeten Trainingsdaten können aus vielen Gründen inkonsistent sein:

- Die Attribute, mit denen die Datensätze in der Datenbank repräsentiert werden, wurden nicht für die KDD-Aufgabe entwickelt.
- Tippfehler oder andere Eingabefehler könnten bei der manuellen Eingabe aufgetreten sein.
- Die manuelle Klassifikation könnte von verschiedenen Personen und/oder zu verschiedenen Eingabezeitpunkten geschehen sein.
- Es kann vorkommen, dass Unterschiede, die zwischen Objekten sichtbar waren, zwischen den zugehörigen Tupeln nicht sichtbar sind. Kann die Klassifikation nicht an den Tupeln selbst vorgenommen werden, so sind diese Datensätze ungeeignet.
- Es besteht die Möglichkeit, dass eine Klassifikation nicht die geeignete Fragestellung für die zugrunde liegenden Daten darstellt.

Inkonsistente Trainingsdaten können je nach Ursache der Inkonsistenz auf verschiedene Weise bereinigt werden, vorausgesetzt die Ursache für die Inkonsistenz ist bekannt. Eine kleine Anzahl von Eingabefehlern kann entweder korrigiert werden oder die Tupel werden aus der Trainingsmenge entfernt. Ist jedoch

eine Vielzahl von Trainingsdaten betroffen, so kann es sein, dass nicht mehr zwischen korrekten und unkorrekten Einträgen unterschieden werden kann. Außerdem könnten viele Einträge genau einer Klasse betroffen sein, wodurch die Trainingsmenge an Ausgewogenheit verlieren wird [FE03].

Die meisten Klassifikationsverfahren setzen konsistente Trainingsdaten voraus. Wie in Kapitel 2.1 anhand des KDD-Prozesses gezeigt, müssen die Daten durch eine geeignete Vorverarbeitung (preprocessing) konsistent gemacht werden.

4 Kombiniertes Data Mining

Nachdem die Grundlagen des KDD, die Data-Mining-Verfahren des Clusterings und der Klassifikation und die Bewertungsmethoden und –kriterien von Klassifikatoren gründlich erläutert worden sind, beschäftigt sich das nächste Unterkapitel mit dem eigentlichen Themenbereich dieser Arbeit, dem „Kombinierten Data Mining“. Dazu muss der Begriff „Kombiniertes Data Mining“ als erstes definiert werden.

4.1 Definition

Der Begriff „Kombiniertes Data Mining“ beschreibt ein Verfahren, bei dem mehrere Data-Mining-Methoden so hintereinander ausgeführt werden, dass die nachfolgenden Verfahren vom ersten profitieren können. In diesem Zusammenhang wird „profitieren“ so verstanden, dass Zwischen- und Endergebnisse, sog. Hilfsinformationen, des ersten Algorithmus in die Ausführung eines nachfolgenden Algorithmus mit einfließen [GM04]. Beispielsweise kann ein Clusteringverfahren Hilfsinformationen zur Laufzeit generieren, die in einem nachfolgenden Klassifikationsverfahren zur Steigerung der Klassifikationsqualität eingesetzt werden können.
Der erste Teil dieses kombinierten Verfahrens besteht darin, die Hilfsinformationen möglichst effizient während der Ausführung des vorgelagerten Data-Mining-Verfahrens zu identifizieren und/oder zu berechnen (vgl. [SK04]).
Der zweite Teil, der gleichzeitig den Inhalt dieser Arbeit darstellt, beschäftigt sich damit, die vorher erfassten Hilfsinformationen in einem nachfolgenden Verfahren im Hinblick auf eine Qualitätssteigerung zu verwerten (siehe Kapitel 3.2 für die Qualitätskriterien von Klassifikatoren). Die Qualität des Endergebnisses steht dabei im Vordergrund. Es ist jedoch durchaus denkbar und wünschenswert, den gesamten kombinierten Prozess durch den Einsatz dieser zusätzlichen Daten effizienter zu gestalten als wenn die Verfahren unabhängig voneinander hintereinander ausgeführt werden würden. Die identifizierten Arten von Hilfsinformationen werden gesondert in Kapitel 5.1 beschrieben.

Data Mining befasst sich mit sehr großen Datenbeständen. Große Mengen an Daten führen im Normalfall dazu, dass der Ressourcenverbrauch an Rechnerzeit und Speicher überproportional in die Höhe schnellt. Durch die Anwendung von speziellen Data-Minining-Methoden und insbesondere des „Kombinierten Data Mining“ soll einerseits eine Reduzierung des Ressourcenbedarfs erreicht werden, andererseits soll versucht werden, die Qualität der Ergebnisse der nachfolgenden Schritte zu steigern.

Im Folgenden werden nun die verschieden Arten des „Kombinierten Data Mining“ angeführt, wie sie in [SK04] definiert wurden:

- **Naives „Kombiniertes Data Mining“**
  Beim naiven „Kombinierten Data Mining“ nutzt das Nachfolgerverfahren das Ergebnis des Vorgängerverfahren. Zum Beispiel kann ein Klassifikationsverfahren die Ergebnisse eines Clusteringverfahren als Trainingsdaten verwenden. In diesem Fall erfolgt eine isolierte Betrachtung der beteiligten Verfahren. Die Kombination der Verfahren führt in keinem Fall zu einer Optimierung.

- **Vorgängerverfahren kennt Nachfolgerverfahren**
  Das Vorgängerverfahren berechnet Hilfsinformationen für das Nachfolgeverfahren. Hilfsinformationen können zum Beispiel arithmetische Werte wie Summen, Minima oder Maxima der verwendeten Daten sein. Dadurch können die Berechnungen des Nachfolgerverfahrens vereinfacht werden.

- **Nachfolgerverfahren kennt Vorgängerverfahren**
  Das Nachfolgeverfahren kann durch die Kenntnis über das Vorgängerverfahren von einem vereinfachten Problem ausgehen. Wenn, zum Beispiel, bei einem Clusteringverfahren nur konvexe Cluster erzeugt werden, muss der nachfolgende Klassifikationsalgorithmus die Attribute nach denen geclustert wurde, nicht mehr berücksichtigen, da das Klassifikationsergebnis sich auf direktem Wege errechnen lässt. In diesem Fall trennt die

mittelsenkrechte Ebene zwischen den Cluster-Mittelpunkten die beiden Cluster optimal.

Zum naiven Kombinierten Data Mining existieren bereits einige Anwendungen. Die Arten „Vorgängerverfahren kennt Nachfolgerverfahren“ und „Nachfolgerverfahren kennt Vorgängerverfahren“ wurden in bis jetzt veröffentlichten Arbeiten nur in der parallelen Arbeit von Klaus Stöttinger [SK04] erwähnt. Abbildung 12 illustriert graphisch den Zusammenhang der Data-Mining-Verfahren bei Kombiniertem Data Mining mit Clustering und Klassifikation.

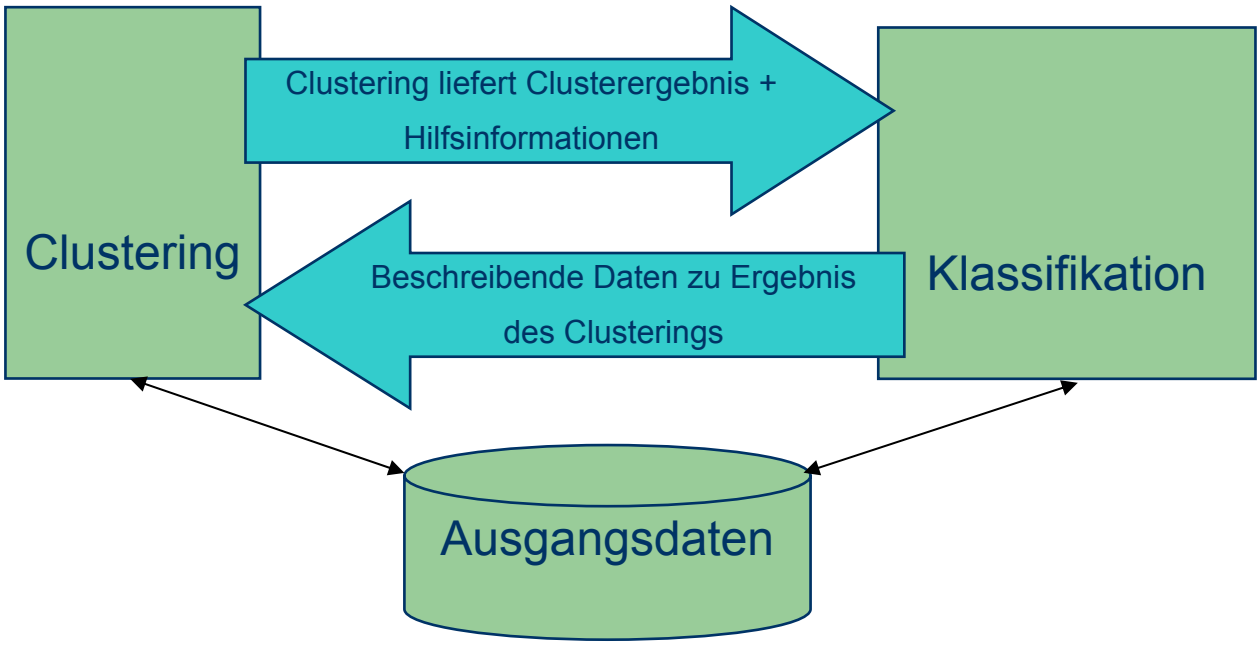


Abbildung 12: Kombiniertes Data Mining mit Clustering und Klassifikation [GM04]

Darüber hinaus soll an dieser Stelle noch eine vierte Form des Kombinierten Data Minings eingeführt werden, bei der ein Verfahren sich der Methodik eines anderen Verfahrens bedient.

- **implizites „Kombiniertes Data Mining“**
  Beim impliziten kombinierten Data Mining verwendet ein Data-Mining-Verfahren zusätzlich zu seinen eigenen Methoden die Methoden und Techniken eines anderen Verfahrens, d.h. dass ein bestimmtes Verfahren implizit in den Prozess der Ergebnisfindung eines anderen Verfahrens eingebunden wird.

Das nun folgende Kapitel diskutiert bestehende Verfahren des kombinierten Data Minings.

4.2 Bestehende Verfahren

In [SK04] sind bereits bestehende Verfahren des „naiven Kombinierten Data Minings“ besprochen worden. Es handelt sich um Verfahren, die einerseits Clustering und Klassifikation, andererseits Clustering und Assoziationsregeln kombiniert verwenden. Werden Clustering und Klassifikation kombiniert angewandt, so enthält das Clusteringergebnis die Klassenzugehörigkeiten, die bei der Klassifikation a priori bekannt sein müssen. Darüber hinaus liefert die Klassifikation ein Erklärungsmodell (z.B. Entscheidungsbaum), das zum besseren Verständnis des Clusteringergebnisses beitragen soll.
Assoziationsregeln drücken Zusammenhänge innerhalb von Transaktionen aus [ES00]. Eine Transaktion ist z.B. eine Kundentransaktion (Warenkorb) im Supermarkt, die alle gekauften Artikel, sog. Items, eines einzelnen Kunden beinhaltet. Eine korrespondierende Assoziationsregel, die häufig gemeinsam gekaufte Artikel enthält, könnte in der Form „{Bier, Cola} → {Chips}“ dargestellt werden. Diese Regel gilt genau dann, wenn in allen Warenkörben, die Bier und Cola enthalten, häufig auch Chips enthalten sind. Eine Kombination von Clustering und Assoziationsregeln ist sinnvoll, um das Assoziationsergebnis, nach dem Erzeugen einer großen Regelmenge, auf die wesentlichen Regeln zu reduzieren.

Die Verfahren des naiven „Kombinierten Data Minings“ werden nachfolgend kurz umrissen. Des Weiteren wird auch ein Ansatz angeführt, der dem im letzten Kapitel neu eingeführten impliziten „Kombinierten Data Mining“ entspricht.

4.2.1 Naives „Kombiniertes Data Mining“

Einige Möglichkeiten **Clustering und Klassifikation** im Sinne des naiven „Kombinierten Data Minings“ anzuwenden sind nachfolgend erläutert:

- **Clustering von Wörtern für Textklassifikation**
  Bei der Klassifikation von Texten wird ein neues Dokument anhand von Dokumentvektoren klassifiziert und jeder Vektor besteht aus den Wörtern eines der vorhandenen Dokumente. Das ist insofern problematisch, da ein Dokument im Normalfall aus mehreren Tausend Wörtern besteht, wobei bei der Klassifikation jedes Wort eine Dimension darstellt, was die Laufzeit enorm steigert [ES00]. Ein vorher durchgeführtes Clustering soll die Wörter in Gruppen einteilen und somit die Zahl der Dimensionen reduzieren. Anwendungen der Kombination von Clustering und Klassifikation von Texten können unter [BM98] und [DKM02] gefunden werden.

- **CBC – Clustering Based (Text) Classification**
  Ist die Anzahl der Dokumente einer Klasse in der Trainingsmenge sehr gering, so hat dies einen negativen Einfluss auf die Klassifikationsgenauigkeit bei der Textklassifikation. Der Ansatz der CBC unterteilt in einem ersten Schritt alle Dokumente in Klassen, unabhängig davon, ob sie bereits einer Klasse zugeordnet sind oder nicht. Dokumenten ohne Klassenzugehörigkeit wird anhand der Dokumente mit Klassenzugehörigkeit eine Klasse zugewiesen. Dadurch kann die Trainingsmenge um die neu klassifizierten Dokumente erweitert und eine höhere Klassifikationsgenauigkeit erwartet werden.

- **Kombination von Self-Organizing Maps und dem K-Means Clustering zur Online-Klassifikation von Sensordaten**
  Hauptpunkt bei dieser Art der Kombination ist die Identifikation und Klassifikation von Kontextinformationen. Die Kontextinformationen (Aufenthaltsort, Aktivität des Benutzers, Status der Applikation, …) werden durch Sensoren identifiziert und müssen dann mithilfe eines Klassifikationsalgorithmus richtig interpretiert und zugeordnet werden. Als Klassifikationsalgorithmus wird die Kohonen Self-Organizing Map (KSOM) [KO94] verwendet. KSOM ist ein neuronales Netzwerk, dass aus den eingehenden Sensordaten eine 2D-Karte erzeugt Da KSOM dazu neigt bereits erlernte Kontexte zu überschreiben, benutzt man im Vorfeld einen K-Means-Algorithmus, um die Kontextinformationen zu gruppieren. Die

Informationen werden auf ihre Clusterzugehörigkeit überprüft und einem Kontext zugeordnet.

- **Automatische Generation eines Fuzzy Classification Systems (FCS) mit Hilfe einer Fuzzy Clustering Methode**
  Für die Anwendung von FCS muss Expertenwissen hinsichtlich der Klassifikationsaufgabe vorhanden sein. Des Weiteren können FCS im Normalfall nicht durch Trainingsdaten erzeugt werden [GG94]. Aus diesem Grund werden sog. Fuzzy Clustering Methoden angewandt, die Objekte einer Klasse zuordnen und darüber hinaus die Wahrscheinlichkeit angeben, dass ein Objekt einer Klasse angehört. Das Fuzzy Clustering soll den Einsatz von Expertenwissen ersetzen.

Nachfolgend werden bestehende Anwendungen der Kombination von **Clustering und Assoziationsregeln** erläutert. Dabei existieren Ansätze, die einerseits Assoziationsregeln aus einem zuvor ausgeführten Clustering erzeugen, andererseits ein Clustering auf vorher identifizierte Assoziationen durchführen:

- **Clustering basierend auf Hypergraphs von Assoziationsregeln**
  Dieser Ansatz beschreibt das Ausführen eines Clusteringalgorithmus auf die häufig auftretenden Items (Frequent Item Sets), die durch den Assoziationsalgorithmus identifiziert wurden. Aus diesen Items wird ein sog. Hypergraph erzeugt, der mithilfe eines partitionierenden Clusteringalgorithmus gruppiert wird. So können anhand dieses Clusterings Items oder ganze Transaktionen klassifiziert werden [HKKM97].

- **Kombination von Clustering und Assoziation für Zielgerichtetes Marketing im Internet**
  In einem Ansatz von Lai und Yang [LY00] wird die Kombination von Clustering und Assoziation in zwei Schritten vollzogen. Im ersten Schritt wird ein Clustering auf die vorhanden Kundendaten durchgeführt, wobei demographische, geographische und auch Daten zum Verhalten der Kunden herangezogen werden. In einem zweiten Schritt wird die Assoziation auf jeden Cluster ausgeführt, die als Ergebnis Assoziationsregeln für jeden Cluster

liefert. Basierend auf diesen Informationen kann Zielgerichtetes Marketing durchgeführt werden.

- **ARCS – Association Rule Clustering System**
  Bei ARCS werden nur Assoziationsregeln im 2-dimensionalen Raum betrachtet, d.h. dass maximal zwei Attribute auf der linken Seite der Regel erlaubt sind [LSW97]. Der Benutzer wählt ein drittes Attribut aus, nach dem anschließend geclustert wird. Als Ergebnis erhält man die „zusammengefassten" Assoziationsregeln. Anschließend werden die Regeln auf ihre Genauigkeit hin überprüft. Sollte Die Genauigkeit ein gewisses Maß unterschreiten, so wird die Assoziation mit geänderten Parametern neu gestartet.

Eine detailliertere Zusammenfassung der oben genannten Anwendungen ist in [SK04] zu finden.

4.2.2 Implizites „Kombiniertes Data Mining" – „Clustering durch Entscheidungsbaumkonstruktion"

Clusteringverfahren benötigen im Gegensatz zu Klassifikationsverfahren keine a priori bekannten Klassen, um Daten gruppieren zu können. Im Normalfall gruppieren Clusteringverfahren Datenpunkte anhand von Distanzfunktionen bzw. Kompaktheitsmaßen. Eine weitere Möglichkeit, ähnliche Datenpunkte zu gruppieren wird in [LXY00] vorgestellt. Anstatt die Ähnlichkeit von Wertepaaren mittels der Distanz der Punkte zu bestimmen, werden Cluster identifiziert, indem Bereiche, die keine Datenpunkte enthalten, von Bereichen mit Datenpunkten getrennt werden. Diese binäre Trennung erfolgt mithilfe eines Entscheidungsbaum-Algorithmus, dem sog. CLTree oder „cluster tree".
Um einzelne Bereiche als „leer" zu kennzeichnen wird der gesamte Datenbereich mit einem besonderen Typ von Datenpunkten gleichmäßig angereichert, den sog. „non existing points" oder N-points (N-Punkte), die einer Klasse N zugeordnet werden. Alle „echten" Datenpunkte werden der Klasse Y zugeordnet (Y-Punkte). Der CLTree ist so in der Lage jene Bereiche, die mehr Y-Punkte als N-Punkte beinhalten, als

Cluster zu identifizieren. Die Abbildungen 13 und 14 illustrieren diese Vorgehensweise.

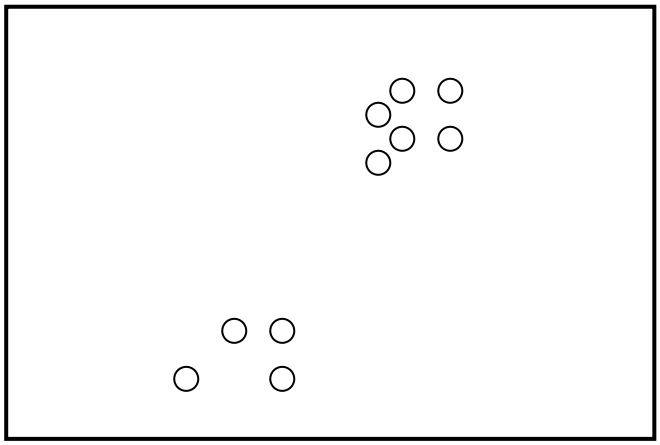


Abbildung 13: Ausgangsdatenbereich für die Anwendung eines CLTrees

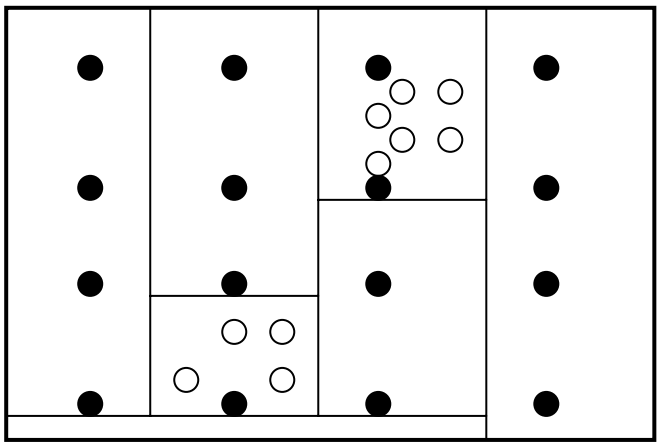


Abbildung 14: Klassifizierter, mit N-points angereicherter Datenbereich

Es ist jedoch auch möglich, die Daten, ohne eine Anreicherung des gesamten Datenbereiches mit N-Punkten, zu gruppieren. Jeder einzelne N-Punkt vergrößert die Datenmenge und daraus resultierend die Laufzeit des Algorithmus. Daher wurde ein Ansatz zur Konstruktion eines CLTrees entwickelt, bei dem die N-Punkte nicht physisch hinzugefügt werden, sondern eine bestimmte Anzahl von N-Punkten für den Ausgangsdatenbereich ermittelt und angenommen wird.

Die Konstruktion eines solchen CLTrees gliedert sich in zwei Phasen:

1. Bei der Erstellung des modifizierten Entscheidungsbaumes wird eine neue Split-Funktion verwendet, um die Daten ohne vorheriges Verteilungswissen zu gruppieren.

2. Nach der Erstellung des Baumes wird ein Pruningschritt durchgeführt, der den Baum beschneidet, sodass brauchbare Cluster als Hyperebenen entstehen.

Vor der Konstruktion des CLTrees muss die Anzahl der „imaginären“ N-Punkte bestimmt werden. Dies geschieht durch die Befolgung eines einfachen Regelwerkes in Abhängigkeit der Y- und N-Punkte. Für einen Knoten K bestimmt man die Anzahl der N-Punkte folgendermaßen:

IF Anzahl der geerbten N-Punkte vom Vaterknoten von E < Anzahl der Y-Punkte von E **THEN**
Anzahl der N-Punkte von E = Anzahl der Y-Punkte von E
ELSE
Anzahl der N-Punkte von E = Anzahl der geerbten N-Punkte vom Vaterknoten von E

In Abbildung 15 wird die Anwendung dieser Regel dargestellt. Ausgehend vom Vaterknoten V, der 1000 Y-Punkte und 1000 N-Punkte (der Vaterknoten erbt 0 N-Punkte) enthält, wird ein Split in die Knoten L (20 Y-Punkte und 500 N-Punkte) und R (980 Y-Punkte und 500 N-Punkte) vorgenommen. Unter Anwendung der obigen Regel wird die Anzahl der N-Punkte von R auf 980 erhöht.

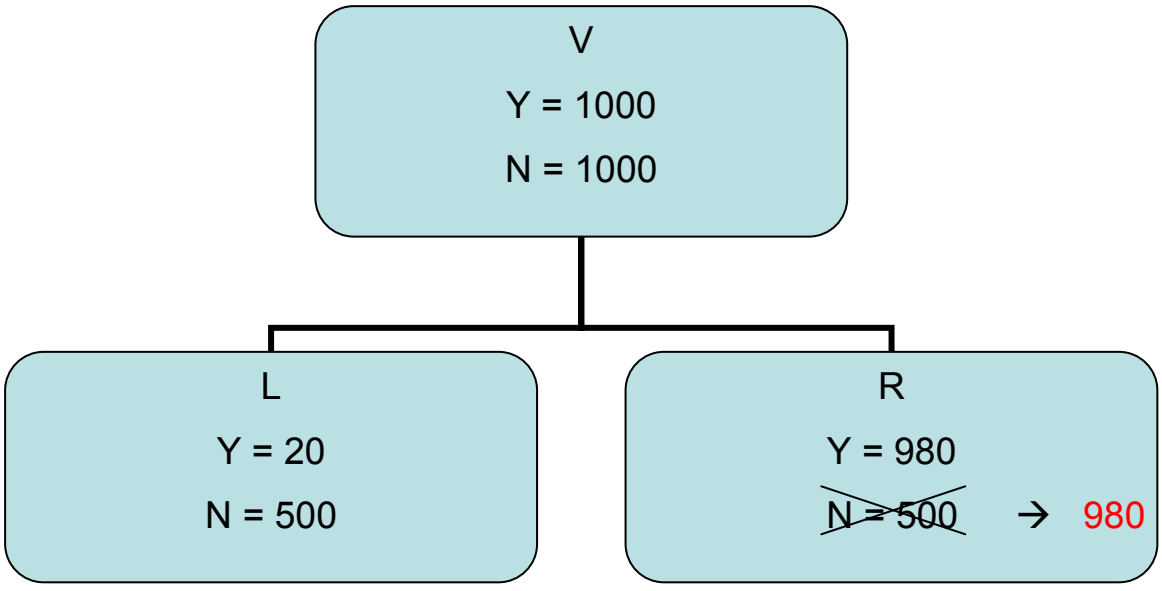


Abbildung 15: Bestimmen der N-Punkte

Damit die N-Punkte nicht physisch hinzugefügt werden müssen, ist es notwendig den Entscheidungsbaum-Algorithmus so zu verändern, dass die N-Punkte einer Partition zur Laufzeit berechnet werden können. Wenn man annimmt, dass die N-Punkte gleichmäßig verteilt sind, dann ist es möglich, für einen binären Split die Verteilung der N-Punkte auf die neuen Partitionen zu berechnen. Wenn z.B. für eine Partition mit 25 Y-Punkten und 25 N-Punkten in einem Wertebereich von 0 bis 10 bei 4 ein möglicher Split evaluiert wird, dann ergibt sich die Anzahl der N-Punkte auf der linken Seite aus $25 \cdot 4/10 = 10$. Die Anzahl der N-Punkte auf der rechten Seite ist $25 – 10 = 15$. Anhand dieser Werte und der Information über die Verteilung der Y-Punkte auf die Partitionen ist es möglich, den Informationsgewinn für diesen Split zu berechnen.
Weiters muss der Algorithmus fähig sein, Splits nicht nur auf einer Seite der Datenpunkte zu erzeugen (vgl. Abbildung 16), d.h. es muss sowohl möglich sein einen Split der Form ((x ≤ Splitwert) OR (x > Splitwert)) zu erzeugen als auch einen Split der Form ((x < Splitwert) OR (x ≥ Splitwert)).

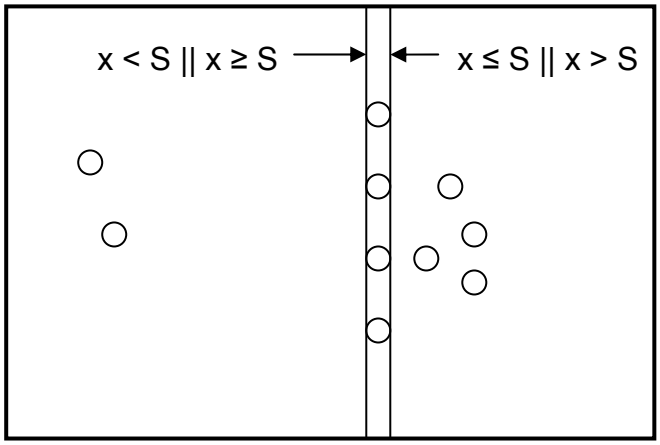


Abbildung 16: Split auf beiden Seiten der Daten

Die „besten“ Splits (z.B. maximaler Informationsgewinn) wie sie bei der Klassifikation erzeugt werden, sind beim Clustering nicht immer optimal. Ein für die Klassifikation optimaler Split tendiert dazu, die Cluster durchzuschneiden, was dazu führt, dass Datenpunkte in den Clustern verloren gehen. Außerdem wird nach der Identifikation des Splitpunktes nicht weiter nach einem besseren Split gesucht.
Diese Problematik wird durch die Verwendung eines neuen Splitkriteriums, dem sog. „lookahead gain criterion“, gelöst. Ausgehend vom Informationsgewinn wird nach solchen Splitpunkten gesucht, die weniger in die Cluster hinein bzw. durch die

Cluster hindurch schneiden. Des Weiteren wird in jeder Dimension nach jenem Datenbereich gesucht, der die wenigsten Datenpunkte enthält, um dadurch die Cluster zu trennen. Es werden „dichte“ von „leeren“ Regionen getrennt.

Die Vorteile des Clusterings durch Entscheidungsbaumkonstruktion gegenüber herkömmlichen Verfahren sind unter anderen:

- Es muss vorher kein Parameter zur Anzahl der Cluster (z.B. K-Means) angegeben werden, da der Algorithmus dichte von leeren Regionen trennt.
- Die gefundenen Cluster werden in Form von Hyperebenen dargestellt, wodurch die Information über die leeren Regionen erhalten bleibt.
- Ausreißer können einfach identifiziert werden, da sie im Normalfall in relativ leeren Regionen liegen.

5 Algorithmen

Im Folgenden werden jene Algorithmen und Frameworks beschrieben, die als Grundlage für die Implementierung der zur Evaluierung der Hilfsinformation benötigten Algorithmen dienen. Dazu wird kurz auf die Vorgehensweise und das Ergebnis des Clusteringalgorithmus „K-Means“ eingegangen, da ein für die Identifikation von Hilfsinformationen optimierter k-means-Algorithmus Trainingsdaten, Testdaten und Hilfsinformationen für den im Zuge dieser Arbeit implementierten Klassifikationsalgorithmus liefert.
Zum Thema Entscheidungsbaum-Klassifikation werden die Algorithmen SLIQ, SPRINT und RainForest vorgestellt, wobei letzterer genauer betrachtet wird, da er als Basis für den hier implementierten Decision-Tree-Klassifikator dient. Das in Kapitel 2.3.3 vorgestellte Verfahren geht davon aus, dass die Datenbank klein genug ist, um vollständig im Hauptspeicher gehalten werden zu können. Heutige Datenbanken entsprechen jedoch in den seltensten Fällen dieser Anforderung [ES00]. Daher ist es notwendig, Entscheidungsbaum-Klassifikatoren zu generieren, die auf Datenbanken beliebiger Größe angewandt werden können.

5.1 K-Means

Das von McQueen 1967 [MQ67] vorgestellte **partitionierende Clusteringverfahren** K-Means ist eine optimierte Variante des „Clustering durch Varianzminimierung“ und das bekannteste und am häufigsten angewendete partitionierende Verfahren [ES00]. Ziel von K-Means ist es, eine gegebene Menge von Datenpunkten in k Klassen zu unterteilen, sodass „ähnliche“ Punkte in Clustern gruppiert werden und Punkten anderer Cluster möglichst „unähnlich“ sind. Die Anzahl der Klassen k wird vom Benutzer vorgegeben. Die Datenpunkte werden zu Beginn zufällig in die k Klassen unterteilt (initiale Zerlegung). Das geschieht, indem zufällig k Punkte als Mittelpunkte (Centroide) dieser Klassen gewählt werden und die übrigen Punkte dem „ähnlichsten“ Centroiden zugewiesen werden. Der Abstand eines Punktes zu einem Centroiden wird als Maß für die Ähnlichkeit verwendet und mittels einer Distanzfunktion berechnet. Die Centroide berechnen sich wie folgt:

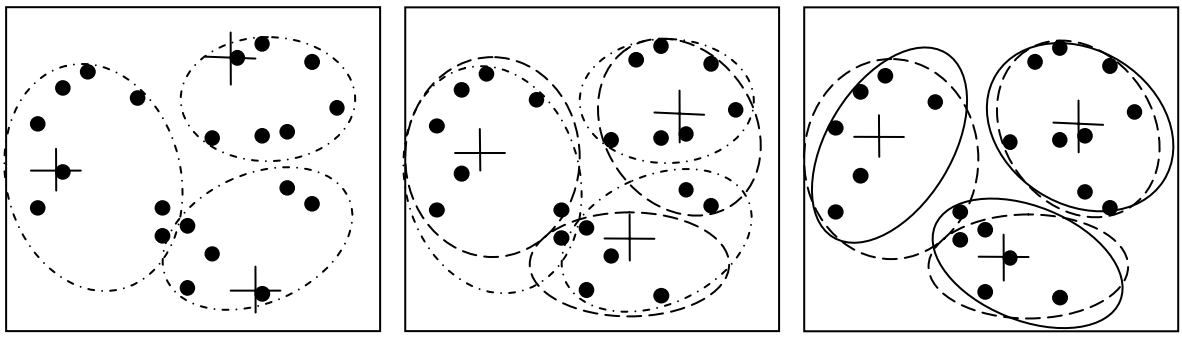
$\mu_C = (\bar{x}_1(C), \bar{x}_2(C), ..., \bar{x}_d(C))$, wobei $\bar{x}_j(C) = \frac{1}{n_C} \cdot \sum_{p \in C} x_j^p$ der Mittelwert der j-ten Dimension aller Punkte in C ist; n_C ist die Anzahl der Objekte in C.

Jeder Datenpunkt wird einzeln einem Centroiden zugeteilt. Ändert sich die Clusterzugehörigkeit eines Punktes, so wird der Centroid des Clusters neu berechnet. Dies geschieht so lange bis sich die Clusterzugehörigkeit keines Punktes mehr ändert. Die Vorgehensweise beim K-Means-Clustering kann als Algorithmus folgendermaßen dargestellt werden (vgl. Abbildung 17):

Kmeans (Punktemenge D, Klassen k)
    Wähle zufällig k Punkte aus D als initiale Centroide
    Repeat
        Ordne die übrigen Punkte aus D ihrem ähnlichsten Centroiden zu
        If Zuordnung der Punkte geändert **Then**
            Ermittle Centroide der geänderten Cluster neu
    Until keine Änderung der Zuordnung mehr

Abbildung 17: K-Means-Algorithmus in Pseudocode

Das K-Means-Clustering liefert konvexe und disjunkte Cluster. Das Ergebnis soll anhand eines Beispiels verdeutlicht werden. Die Abbildungen 18 (a) bis (c) illustrieren die Ergebnisfindung bei K-Means.



(a) (b) (c)

Abbildung 18: Beispiel für K-Means Algorithmus

Ausgehend von der Punktemenge in Abbildung 18 (a) werden zufällig drei (k = 3) Centroide (in den Abbildungen als Kreuze dargestellt) ausgewählt. Die übrigen

Punkte werden ihrem nächsten Centroiden zugeordnet. Abbildung 18 (a) zeigt die Cluster nach der 1. Iteration als gestrichelte Ellipsen.
In Abbildung 18 (b) ist die Neuzuordnung der Datenpunkte nach der Berechnung der neuen Clusterzentren gut zu erkennen. Nach einer wiederholten Neuberechnung der Centroide sind in Abbildung 18 (c) die endgültigen Cluster als durchgezogene Ellipsen dargestellt.

Es gibt Methoden und Heuristiken, um die Initialisierung von K-Means zu verbessern [ES00]. Optimierungen hinsichtlich der effizienten Identifikation von Hilfsinformationen sind ausführlich in [SK04] beschrieben.

5.2 SLIQ

Der Entscheidungsbaum-Algorithmus SLIQ aus [MAR96] ist für den Einsatz mit großen Datenbanken skaliert und eignet sich sowohl für numerische als auch kategorische Attribute. Er erzeugt binäre Splits und verwendet **spezielle Datenstrukturen**, die es ermöglichen, dass Attributausprägungen nur einmal zu Beginn der Baumgenerierungsphase sortiert werden müssen. Andere Verfahren müssen im Normalfall die Daten für die Erzeugung eines neuen Knotens immer wieder sortieren.
SLIQ verwendet im Gegensatz zu anderen Verfahren nicht den Informationsgewinn als Split-Kriterium, sondern berechnet den sog. „Gini-Index“. Der **Gini-Index** ist genau wie der Informationsgewinn ein Maß für die Unreinheit von Partitionen. Für eine Menge T von Trainingsobjekten und den relativen Häufigkeiten p_i der Klassen c_i in T ist der Gini-Index definiert als

$$gini(T) = 1 - \sum_{i=1}^{k} p_i^2 \quad \text{wobei } gini(T) \in [0,1] \text{ gilt.} \tag{5.2}$$

Der Gini-Index für die Partitionen T_1, T_2, …, T_m von T ist definiert als

$$gini(T_1, T_2, ..., T_m) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i). \tag{5.3}$$

Minimale Unreinheit liegt dann vor, wenn für k = 2 Klassen, eine Partition nur Elemente einer Klasse enthält. Maximale Unreinheit liegt dann vor, wenn für k = 2 Klasse, beide Partitionen jeweils genau die Hälfte der Elemente enthalten.

Bevor ein Split-Algorithmus von SLIQ eingeführt und begriffen werden kann, müssen die zur Unterstützung von großen Datenbeständen hinzugefügten speziellen Datenstrukturen erläutert werden.

- **Attributlisten**
  Eine Attributliste enthält die Attributausprägungen eines Attributs in aufsteigender Reihenfolge sortiert. Darüber hinaus ist zu jeder Ausprägung eine Referenz auf den korrespondierenden Eintrag in der Klassenliste vorhanden.

- **Klassenliste**
  In der Klassenliste ist jeder Trainingsdatensatz in Form seiner ID, seiner Klasse und einem Verweis auf den Blattknoten des Entscheidungsbaumes, zu dessen Partition der Datensatz gehört, gespeichert.

- **Histogramme**
  Die Histogramme werden zu Evaluierung der potentiellen Splitpunkte verwendet. Sie enthalten die Häufigkeiten der einzelnen Klassen am jeweiligen Blattknoten.

Nach [ES00] existiert ein Algorithmus wie in der folgenden Abbildung 19:

```
EvaluiereSplits (Entscheidungsbaum B)
For each Attribut A do
        For each Wert W in der Attributliste von A do
                Bestimme den zugehörigen Eintrag E der Klassenliste;
                K = zugehöriger Blattknoten in B von E;
                Aktualisiere das Histogramm von K;
                If A = numerisches Attribut Then
                        Berechne Gini-Index für die Partitionen von K durch den Test A ≤ W;
                If A = kategorisches Attribut Then
```

For each Knoten K von B **do**

Bestimme jene Teilmenge der Attributwerte von A mit dem kleinsten Gini-Index;

Abbildung 19: SLIQ in Pseudocode

Abschließend soll die Vorgehensweise von SLIQ anhand eines Beispiels verdeutlicht werden. Als Trainingsdaten fungieren die Tupel der Tabelle 3 aus Kapitel 2.3. Diese Daten sind hier aus Gründen der Übersichtlichkeit in Form der Tabelle 8 noch einmal angeführt.

ID	Alter	Familienstand	Kinder
1	23	geschieden	nein
2	17	ledig	nein
3	43	ledig	nein
4	68	geschieden	ja
5	32	verheiratet	ja

Tabelle 8: Trainingstabelle für SLIQ

Die Tabellen 9 (a) und (b) zeigen die aus den Trainingsdaten erzeugten Attributlisten.

Alter	ID
17	2
23	1
32	5
43	3
68	4

(a)

Familienstand	ID
geschieden	1
geschieden	4
verheiratet	5
ledig	2
ledig	3

(b)

Tabelle 9: Attributlisten für SLIQ

Für die potentiellen Splitpunkte eines jeden Attributs werden Histogramme erstellt, aus denen der Gini-Index und somit die Reinheit der entstehenden Partitionen berechnet werden kann. Tabelle 10 zeigt den Endzustand eines Histogramms für den Knoten N1 bezüglich des Attributs „Familienstand“. Da SLIQ nur binäre Splits durchführt, wurden als Teilmengen mit dem kleinsten Gini-Index „verheiratet“ und „¬verheiratet“ (ledig oder geschieden) identifiziert.

	Kinder = nein	Kinder = ja
verheiratet	0	1
¬verheiratet	3	1

Tabelle 10: Histogramm für SLIQ

Aus dem Histogramm lässt sich der Gini-Index für diese Partitionierung einfach berechnen. Der Gini-Index für die gesamte Trainingsmenge T aus Tabelle 8 beträgt durch Anwendung der Formel aus (5.2) gini(T) = 0,48. Der Gini-Index für die Partitionierung ($T_{verheiratet}, T_{\neg verheiratet}$) berechnet sich nach (5.3) wie folgt:

$$gini(T_{verheiratet}, T_{\neg verheiratet}) = (\frac{1}{5} \cdot (1 - ((\frac{0}{1})^2 + (\frac{1}{1})^2))) + (\frac{4}{5} \cdot (1 - ((\frac{3}{4})^2 + (\frac{1}{4})^2))) =$$

$$gini(T_{verheiratet}, T_{\neg verheiratet}) = (\frac{1}{5} \cdot 0) + (\frac{4}{5} \cdot 0{,}375) = \underline{\mathbf{0{,}3}}$$

Die Zugehörigkeit eines Datensatzes zu einem Blattknoten wird in der Klassenliste gespeichert. Die Klassenlisten vor und nach dem ersten Split und die dazugehörigen Entscheidungsbäume darunter sind in Abbildung 20 ersichtlich.

ID	Klasse	Blattknoten
1	nein	N1
2	nein	N1
3	nein	N1
4	ja	N1
5	ja	N1

ID	Klasse	Blattknoten
1	nein	N3
2	nein	N3
3	nein	N3
4	ja	N3
5	ja	N2

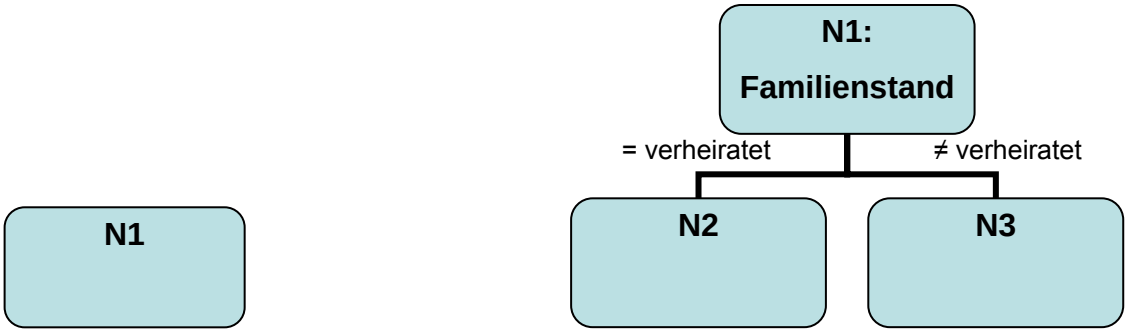


Abbildung 20: Klassenlisten und Entscheidungsbäume bei SLIQ

SLIQ hält die Klassenlisten im Hauptspeicher unter der Annahme, dass die Menge der Trainingsdaten dies zulässt. Eine Weiterentwicklung von SLIQ ist der Algorithmus SPRINT, bei dem auf eine speicherresistente Repräsentation der Klassenlisten verzichtet wird, um völlig unabhängig vom Hauptspeicher operieren zu können.

5.3 SPRINT

Der Algorithmus SPRINT [SAM96] stellt eine Optimierung von SLIQ dar. Während SLIQ darauf angewiesen ist, dass genügend Hauptspeicher für die Klassenlisten zur Verfügung steht, verzichtet SPRINT auf hauptspeicherresistente Datenstrukturen. Daher ist SPRINT für die Anwendung auf beliebig große Datenbanken geeignet. Anstelle von speicherresistenten Klassenlisten wird in die Attributlisten, die auf dem Sekundärspeicher liegen, ein zusätzliches Attribut „Klasse“ eingefügt. Die Unabhängigkeit vom Hauptspeicher geht auf Kosten von Redundanz und der Vergrößerung der Attributlisten. Außerdem bildet SPRINT im Gegensatz zu SLIQ nicht eine Attributliste pro Attribut der Trainingsdaten, sondern jeweils für jeden Knoten des Entscheidungsbaumes. Die Attributlisten des Wurzelknotens, die so wie bei SLIQ sortiert sind, werden für die Sohnknoten in die entsprechenden Partitionen zerlegt und müssen nicht wieder sortiert werden.
Darüber hinaus ist SPRINT relativ gut für eine parallele Verarbeitung geeignet, da die Attributlisten nicht zentral, d.h. genau eine Attributliste pro Attribut der Trainingsmenge, sondern dezentral, d.h. eine Partition der Attributliste pro Knoten, verwaltet werden. Die Tabellen 11 (a) und (b) zeigen die Attributlisten von SPRINT für die Trainingsdaten aus Tabelle 8.

Alter	Klasse	ID
17	nein	2
23	nein	1
32	ja	5
43	nein	3
68	ja	4

(a)

Autotyp	Klasse	ID
geschieden	nein	1
ledig	nein	2
geschieden	ja	4
ledig	nein	3
verheiratet	ja	5

(b)

Tabelle 11: Attributlisten bei SPRINT

Aus den Attributlisten lassen sich mithilfe eines sog. Cursors, der die Trainingsdaten sequentiell durchläuft, für **numerische Attribute** die zugehörigen Histogramme erzeugen. Der Cursor positioniert sich jeweils zwischen zwei Datensätze und evaluiert das arithmetische Mittel als Splitpunkt. Bei kategorischen Attributen werden lediglich die Häufigkeiten der Klassenzugehörigkeiten pro Attributausprägung in ein Histogramm eingetragen. Abbildung 21 illustriert beispielhaft die Splits an zwei verschiedenen Cursorpositionen beim Attribut „Alter“.

	Hoch	Niedrig
≤ 27,5	2	0
> 27,5	1	2

Cursorposition 2 – Split zwischen ID = 1 und ID = 5

	Hoch	Niedrig
≤ 55,5	3	1
> 55,5	0	1

Cursorposition 4 – Split zwischen ID = 3 und ID = 4

Abbildung 21: Histogramme für ein numerisches Attribut bei SPRINT

Der Split an Cursorposition 4 teilt die Trainingsdaten in reinere Partitionen. Die Attributlisten und die korrespondierenden Entscheidungsbäume nach dem ersten Split über das Attribut „Alter“ können der Abbildung 22 entnommen werden. Der Split wird über das Attribut „Alter“ durchgeführt, da das Cursor-Konzept nur für numerische Dimensionen angewendet werden kann.

Alter	Klasse	ID
17	nein	2
23	nein	1
32	ja	5
43	nein	3

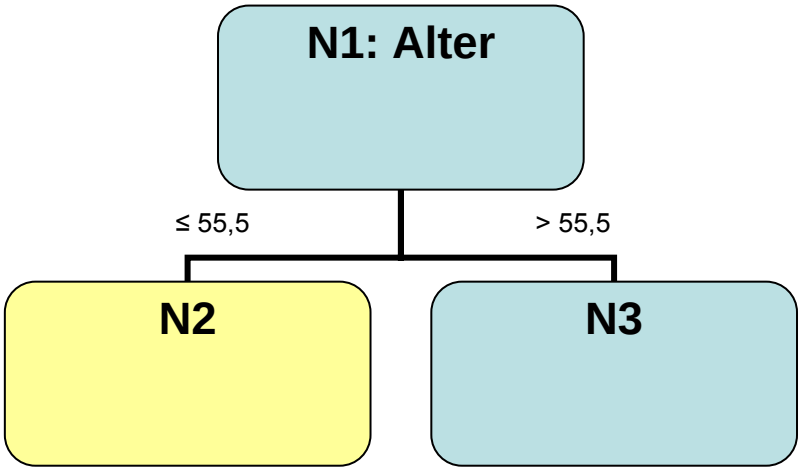
Familienstand	Klasse	ID
geschieden	nein	1
ledig	nein	2
ledig	ja	3
verheiratet	ja	5

Attributlisten für Knoten N2

Alter	Klasse	ID
68	ja	4

Familienstand	Klasse	ID
geschieden	ja	4

Attributlisten für Knoten N3



Baum nach dem ersten Split über das Attribut „Alter“

Abbildung 22: Attributlisten und Entscheidungsbaum nach Split über das Attribut „Alter“

5.4 RainForest

Die oben vorgestellten Entscheidungsbaum-Algorithmen haben jeweils einen Nachteil. Während SLIQ unter der Annahme operiert, dass die Klassenlisten in den Hauptspeicher passen und somit nicht für beliebig große Datenbanken skaliert ist, verwendet SPRINT keine speicherresistenten Datenstrukturen und nutzt den vorhandenen Hauptspeicher nicht aus. Um diese Schwächen zu überwinden, wurde in [RGG98] der generische Ansatz „RainForest“ vorgestellt.
RainForest ist ein Algorithmen-Framework, das keine Vorschriften hinsichtlich einer Splitstrategie macht. Es ist praktisch mit allen vorhandenen Implementierungen von Entscheidungsbaum-Algorithmen anwendbar. RainForest verfügt über eine spezielle speicherresistente Datenstruktur, die den vorhandenen Hauptspeicher effizient ausnutzt [ES00]. Diese Datenstruktur ist die sog. AVC-Gruppe, die aus jeweils einer AVC-Menge (AVC-Set) pro Attribut der Trainingsdaten besteht. AVC ist ein Akronym für „Attribute-Value, Classlable“ (Attribut-Wert, Klasse). Jedes AVC-Set besteht aus Tupeln der Form

$$(a_i, c_j, counter)$$

wobei a_i die Attributausprägung eines Attributes A der Trainingsdaten T darstellt, c_j die Klassenzugehörigkeit des gesamten Datensatzes bzgl. aller Klassen C beschreibt und counter (Zähler) die Anzahl aller gleich lautenden Attributausprägungen a_i ist. Die AVC-Menge eines Attributs enthält für jede Attributausprägung ein Klassenhistogramm, das zur Evaluierung der Splitpunkte herangezogen werden kann. Ein AVC-Set nimmt im schlimmsten Fall die Größe einer entsprechenden Attributliste bei SPRINT an, und zwar genau dann, wenn alle Ausprägungen a_i von A unterschiedliche Werte besitzen. Im besten Fall passen alle AVC-Mengen, d.h. die gesamte AVC-Gruppe eines Knotens, in den Hauptspeicher. Wie die AVC-Gruppe bei der Erzeugung eines Entscheidungsbaums eingesetzt wird, sollen die folgenden Schritte, die für jeden Knoten des Baumes durchgeführt werden müssen, verdeutlichen:

1. Erzeugen der AVC-Gruppe des Knotens
   Aus der Partition des aktuellen Knotens (den gesamten Trainingsdaten im Falle des Wurzelknotens) wird für jedes Attribut ein AVC-Set erzeugt.

2. Auswählen des Splitattributs und des Splitwertes
   Anhand eines Reinheitskriteriums (Informationsgewinn, Gini-Index) wird jedes AVC-Set auf den optimalen Split hin untersucht.

3. Erzeugen von Sohn-Partitionen
   Basierend auf den evaluierten Splitwert wird die Partition des aktuellen Knotens in Sohn-Partitionen zerlegt, d.h. es wird für jeden Sohn-Knoten eine Partition erzeugt, die das Splitkriterium erfüllt.

4. Rekursives Erzeugen des Baumes entlang aller Sohn-Knoten
   Für jeden Sohn-Knoten wird aus seiner Sohn-Partition die zugehörige AVC-Gruppe gebildet und der Baum rekursiv aufgebaut.

Das folgende Beispiel soll die Erzeugung des Entscheidungsbaumes mithilfe von AVC-Gruppen veranschaulichen. Abbildung 23 zeigt die AVC-Mengen am Wurzelknoten N1 für die Trainingsdaten aus Tabelle 8.

Wert	Klasse	Zähler
17	nein	1
23	nein	1
32	ja	1
43	nein	1
68	ja	1

AVC-Menge Alter für N1

Wert	Klasse	Zähler
geschieden	nein	1
geschieden	ja	1
ledig	nein	2
verheiratet	ja	1

AVC-Menge Familienstand für N1

Abbildung 23: AVC-Sets für den Wurzelknoten N1

Bei einem binären Split über „Familienstand“ mit den Splitkriterien ($a_i \neq$ verheiratet) für Knoten N2 und ($a_i =$ verheiratet) für Knoten N3 ergeben sich die AVC-Sets aus Abbildung 24.

Wert	Klasse	Zähler
17	nein	1
23	nein	1
43	nein	1
68	ja	1

AVC-Menge Alter für N2

Wert	Klasse	Zähler
geschieden	nein	1
geschieden	ja	1
ledig	nein	2

AVC-Menge Familienstand für N2

Wert	Klasse	Zähler
43	ja	1

AVC-Menge Alter für N3

Wert	Klasse	Zähler
verheiratet	ja	1

AVC-Menge Familienstand für N3

Abbildung 24: AVC-Sets für die Knoten N2 und N3

Der **generische Algorithmus** des RainForest-Frameworks besitzt drei Parameter [ES00]. Zu Beginn der Ausführung wird der Algorithmus mit einem Knoten K als Wurzelknoten des Entscheidungsbaumes aufgerufen. Die Menge D fungiert als Trainingsmenge bzw. als jene Teilmenge der Trainingsdaten, die alle Bedingungen von der Wurzel des Entscheidungsbaumes bis zum Knoten K erfüllt. Der dritte Parameter ist der Algorithmus S, der alle Methoden zur Evaluierung potentieller Splits und zur Auswahl des jeweils besten Splits beinhaltet. Der generische Algorithmus gliedert sich unter Verwendung dieser Parameter nach [ES00] wie folgt (vgl. Abbildung 25):

KonstruiereEntscheidungsbaum (Knoten K, Trainingsdaten D, Algorithmus S)
For each Attribut A **do**
    Wende S auf die AVC-Menge von A an, um den besten Split von A zu bestimmen;
Wähle mithilfe von S den insgesamt optimalen Split aus;
Partitioniere D entsprechend dieses Splits in D_1, …, D_m;
Erzeuge k Söhne K_1, …, K_m von K;
For i **from** 1 **to** m **do**
    KonstruiereEntscheidungsbaum (K_i,D_i,S);

Abbildung 25: RainForest-Framework in Pseudocode

In Abhängigkeit vom zur Verfügung stehenden Hauptspeicher, existieren verschiedene Ansätze zur effizienten Abarbeitung von RainForest. Man geht davon aus, dass entweder die gesamte AVC-Gruppe des Wurzelknotens in den Hauptspeicher passt, oder zumindest jede einzelne AVC-Menge speicherresistent gehalten werden kann [ES00]. Im Folgenden werden die Algorithmen zur effizienten Nutzung des Hauptspeichers beschrieben:

- **Algorithmus RF_Write**
  Dieser Ansatz geht davon aus, dass die gesamte AVC-Gruppe des Wurzelknotens im Hauptspeicher abgelegt werden kann. Die Trainingsdatensätze werden pro Knoten zwei Mal gelesen, einmal zur Erstellung der AVC-Gruppe, ein zweites Mal, um die Trainingsmenge aufgrund des besten Splits zu partitionieren.

- **Algorithmus RF_Read**

  Nachdem ab einer bestimmten Anzahl von Knoten die Möglichkeit besteht, dass nicht mehr alle AVC-Gruppen der bereits erzeugten Knoten in den Hauptspeicher geschrieben werden können, verwendet man die Strategie von RF_Read. Es wird völlig darauf verzichtet, die Partitionen von D auf dem Sekundärspeicher zu erstellen. In Abhängigkeit des begrenzten Arbeitsspeichers werden zu den aus der Datenbank selektierten Partitionen so viele AVC-Menge wie möglich erzeugt. Die Ersparnis durch den Verzicht des Schreibens der Partitionen bringt den Nachteil mit sich, dass im Normalfall die Trainingsmenge für jede Ebene des Entscheidungsbaumes mehrmals gelesen werden muss.

- **Algorithmus RF_Hybrid**

  RF_Hybrid ist eine Kombination aus RF_Write und RF_Read. Im einfachsten Fall arbeitet RF_Hybrid so lange nach der Strategie RF_Read, bis die AVC-Gruppen aller bisher erzeugten Knoten nicht mehr in den Hauptspeicher passen. In diesem Fall fährt der Algorithmus nach der Strategie RF_Write fort und erzeugt Partitionen im Sekundärspeicher. Der Vorteil dieser Abarbeitung liegt darin, dass der Hauptspeicher in den höheren Ebenen des Entscheidungsbaumes effizienter genutzt wird, während in den tieferen Ebenen darauf verzichtet wird, die Trainingsdatensätze wie bei RF_Read immer öfter zu lesen.

- **Algorithmus RF_Vertical**

  Dieser Ansatz wurde für den Fall entwickelt, dass die AVC-Gruppe des Wurzelknotens nicht in den Arbeitsspeicher geschrieben werden kann. Die einzelnen AVC-Mengen der Wurzel können jedoch speicherresistent gehalten werden. RF_Vertical arbeitet ähnlich wie RF_Hybrid mit dem Unterschied, dass die Attribute der Trainingsmenge vor der Ausführung in Klassen hinsichtlich der wahrscheinlichen Größe ihrer AVC-Mengen eingeteilt werden. Bei der Evaluierung des besten Splits werden zuerst die Attribute herangezogen, deren AVC-Mengen zusammen in den Hauptspeicher passen. Im Anschluss werden die „großen“ AVC-Mengen einzeln evaluiert. Erst wenn

für jedes Attribut das Splitkriterium berechnet worden ist, wird aus allen Kriterien der beste Split gewählt.

Im Normalfall kann man davon ausgehen, dass die AVC-Gruppe des Wurzelknotens in den Hauptspeicher passt [RGG98]. Der Algorithmus RF_Vertical wird daher nur in Ausnahmefällen zur Anwendung kommen. Der im Zuge dieser Arbeit implementierte Entscheidungsbaum-Klassifikator (siehe Kapitel 7) verwendet RF_Write.

6 Meta- und Hilfsinformationen

Beim kombinierten Data Mining „Nachfolgerverfahren kennt Vorgängerverfahren“ sollen Hilfsinformationen in den nachfolgenden Algorithmus einfließen, die zu einer Qualitätssteigerung des Ergebnisses führen können. Die Hilfsinformationen stammen aus einem für die effiziente Identifizierung und Berechnung von Hilfsinformationen optimierten K-Means-Algorithmus [SK04]. Der nachfolgende Algorithmus ist im vorliegenden Fall ein Klassifikationsalgorithmus, genauer gesagt ein Entscheidungsbaum-Algorithmus. Eine grobe Gliederung der Hilfsinformationen lässt sich folgendermaßen vornehmen:

- **Metainformationen**
  Metainformation ist in trivialer Form vorhanden, da bei der Implementierung des nachfolgenden Algorithmus bekannt ist, welcher Algorithmus vorher ausgeführt wird. Es ist außerdem bekannt, welche Hilfsinformationen der vorher ausgeführte Algorithmus liefern kann. Metainformationen können auch Aufschluss darüber geben, welche Hilfsinformationen das Potential besitzen, Einfluss auf das Endergebnis des Nachfolgers auszuüben.

- **Berechnete Hilfsinformationen**
  Unter „berechneten Hilfsinformationen“ werden alle Hilfsinformationen verstanden, die im vorgelagerten Algorithmus zur Laufzeit als Zwischen- oder Endergebnisse errechnet werden konnten. Die Hilfsinformationen wurden dabei noch keiner Wertung unterzogen, es ist daher nicht bekannt, welchen Einfluss (positiv, negativ, indifferent) sie auf die Qualität des Endergebnisses des Nachfolgers haben.

In den nachfolgenden Unterkapiteln werden die Arten von Hilfsinformationen in Bezug auf die obige Grobgliederung besprochen. Darüber hinaus wird Stellung zum Einflusspotential einer jeden Hilfsinformation genommen.

6.1 Arten von Hilfsinformationen

Dem Nachfolgeralgorithmus ist als Metainformation bekannt, welches Verfahren im ersten Schritt durchgeführt wurde. Des Weiteren ist bekannt, welche Beschaffenheit die Ausgangsdaten für den Klassifikationsalgorithmus (Ergebnis des Clusteringalgorithmus) besitzen. Darüber hinaus gibt das Clusteringergebnis Aufschluss darüber, welche Attribute im Ergebnis enthalten sind und nach welchen Attributen geclustert wurde.
Aus diesen Metainformationen und dem Wissen über die berechneten Hilfsinformationen lässt sich die Menge von Hilfsinformationen in die Klassen „Anzahl bzw. Art der Attribute“, „Lage der Ausgangsdaten“ und „Kennzahlen“ einordnen.

- **Anzahl und Art der Attribute**
  Das K-Means-Clustering eignet sich nicht zur Gruppierung von kategorischen Daten. Daher ist es interessant festzustellen, inwieweit sich die Klassifikationsgenauigkeit und andere Qualitätsmerkmale eines Klassifikators verändern, wenn Daten unterschiedlicher Skalenniveaus verwendet werden. Die Information über die Anzahl und die Art der Attribute ist Metainformation, da sie durch die vorhandenen Trainingsdaten festgeschrieben und unabänderlich ist.

  - Nur numerische Attribute
    Es fließen nur numerische Attribute in die Konstruktion des Entscheidungsbaumes ein, d.h. jene Attribute, nach denen auch geclustert wurde.

  - Numerische und kategorische Attribute
    Zur Erzeugung des Entscheidungsbaumes werden Datensätze verwendet, die eine Kombination aus numerischen und kategorischen Attributen enthalten, d.h. Attribute, nach denen geclustert und nicht geclustert wurde.

  - Kategorische Attribute

Werden nur kategorische Attribute zur Entscheidungsbaum-Konstruktion verwendet, so basiert die Klassifikation zur Gänze auf einer dem Clustering verschiedenen Datenmenge.

- **Lage der Ausgangsdaten**

  Das vorher ausgeführte K-Means-Clustering liefert konvexe Cluster. Als Hilfsinformation erhält der Decision-Tree-Klassifikator Informationen über die Lage der Punkte in den Clustern. Mittels dieser Lageinformation soll versucht werden festzustellen, welcher Typ von Ausgangsdaten sich am besten zur Konstruktion des Entscheidungsbaumes eignet.

  - Alle Tupel

    Alle Tupel des Clusterings fließen in die Konstruktion des Entscheidungsbaumes ein. Diese Vorgehensweise eignet sich vor allem dann, wenn die Ausgangsdatenmenge relativ klein ist und die Bewertung des Klassifikators mittels Kreuzvalidierung erfolgt.

  - Zufallsdaten

    Ausgehend von einer normalverteilten Ausgangsdatenmenge werden geeignet große Stichproben für Trainings- und Testdaten gezogen.

  - Repräsentative Daten

    Als „repräsentative Daten“ fungieren Daten aus dem Clusteringergebnis, die entweder besonders nahe am Zentrum der Cluster liegen oder besonders weit davon entfernt sind. Diese Art der Daten muss vom Clusteringverfahren als Hilfsinformation bereitgestellt werden. Außerdem werden Schwellwerte festgelegt, um die Datenpunkte als nah oder fern bezeichnen zu können.

    Beim K-Means Algorithmus wird bei der Ermittlung dieser Punkte die Distanz der Punkte zu ihrem Centroiden als Maß für die Nähe und die Entfernung herangezogen. Je geringer die Distanz eines Punktes zu seinem Centroiden, desto näher ist dieser Punkt, und umgekehrt. Als Trainingsdaten für den Decision-Tree-Klassifikator dienen zusätzlich

erzeugte Tabellen, welche die nahesten bzw. entferntesten Punkte enthalten

- **Kennzahlen**

  Der K-Means-Algorithmus mit Hilfsinformation berechnet möglichst effizient eine Reihe von Hilfsinformationen in Form von Kennzahlen zu den einzelnen Dimensionen und den identifizierten Clustern.

  - Hilfsinformationen zu den Dimensionen
    - Summe

      Die Summe ist das Ergebnis der Addition aller Ausprägungen einer Dimension.
    - Quadratsumme

      Die Quadratsumme wird zur Berechnung der Standardabweichung einer Dimension verwendet.
    - Mittelwert

      Der Mittelwert oder auch Erwartungswert bildet das arithmetische Mittel einer Dimension.
    - Standardabweichung

      Die Standardabweichung wird mithilfe der Summe, Quadratsumme und der Anzahl der Punkte berechnet.
    - Minimum

      Das Minimum ist der kleinste Wert einer Dimension.
    - Maximum

      Das Maximum ist der größte Wert einer Dimension.
    - Median

      Der Median ist der „Mittelwert“ einer Dimension in Form eines repräsentativen Punktes.
    - 25%-Quantil

      Das 25%-Quantil ist ein repräsentativer Punkt, der das erste Viertel einer Dimension begrenzt.
    - 75%-Quantil

      Das 75%-Quantil ist ein repräsentativer Punkt, der die ersten ¾ einer Dimension begrenzt.

- Anzahl der Punkte

  Die Anzahl der Punkte wird zur Berechnung des Mittelwerts und der Standartabweichung benötigt. Sie ist für alle Dimensionen gleich.

o Hilfsinformationen zu den Clustern

- Summe

  Die Summe ist das Ergebnis der Addition aller Ausprägungen eines Clusters einer bestimmten Dimension.

- Quadratsumme

  Die Quadratsumme wird zur Berechnung der Standardabweichung eines Clusters benötigt.

- Mittelwert

  Der Mittelwert oder auch Erwartungswert bildet das arithmetische Mittel eines Clusters. Er dient zur Berechnung der Verteilung des Clusters.

- Standardabweichung

  Die Standardabweichung wird mithilfe der Summe, Quadratsumme und der Anzahl der Punkte berechnet. Sie wird zur Berechnung der Veiteilung des Clusters benötigt.

- Minimum

  Das Minimum ist der kleinste Wert eines Clusters. Es gibt Aufschluss über die Verteilung eines Clusters.

- Maximum

  Das Maximum ist der größte Wert eines Clusters. Es gibt Aufschluss über die Verteilung eines Clusters.

- Anzahl der Punkte

  Die Anzahl der Punkte wird zur Berechnung des Mittelwerts und der Standartabweichung benötigt. Sie sagt aus, wie viele Punkte einem Cluster angehören.

- Wurzel über die Anzahl der Punkte

  Die Wurzel über die Anzahl der Punkte wird am Ende des Clusteringprozesses berechnet.

6.2 Potential der Hilfsinformationen

Um das Potential der identifizierten Hilfsinformationen hinsichtlich einer Qualitätssteigerung des Klassifikators festzustellen, müssen zuerst jene Daten und Prozesse begriffen werden, die für die Klassifikatorqualität als „kritisch“ zu bezeichnen sind. Kritisch sind jene Daten und Prozesse, die bei ihrer Abarbeitung großen Einfluss auf die Qualität des Klassifikators nehmen.

Bei der Klassifikation geht man von Datensätzen aus, die bereits Klassen zugeordnet sind, um ein Modell zu konstruieren, anhand dessen man zukünftige Daten klassifizieren kann. Geht man, so wie im vorliegenden Fall, von Clusteringergebnissen aus, so dient die Clusterzugeörigkeit als Ausgangsklasse eines Datensatzes. Die Trainingsdaten eines Klassifikators sind eine äußerst kritische Menge in Bezug auf dessen Qualität. Es ist z.B. von großer Wichtigkeit, dass diese Daten homogen über die zugeordneten Klassen (Cluster) verteilt sind, damit jede Klasse repräsentativ in den Trainingsdaten vertreten ist [FR02]. Des Weiteren bestimmt die **Lage der Datenpunkte in den Clustern** zu einem großen Teil die Klassifikationsbedingungen des Klassifikators und somit die Klassen von zukünftigen Daten. Daher ist es sinnvoll, Trainingsdaten zu verwenden, deren Datensätze innerhalb der Klassen bestimmte Lageeigenschaften aufweisen.

Als kritische Prozesse bei der Klassifikator-Konstruktion sind die Identifikation des Splitattributs und das Finden des besten Splits zu bezeichnen. Die Splitpunkte sind abhängig vom Typ der Attribute der Trainingsmenge (numerisch oder kategorisch) und von der Verteilung der Werte innerhalb eines Attributs. Bei der Entwicklung einer (neuen) Splitstrategie sind im Besonderen Verteilungsinformationen von großer Bedeutung. Sie können einerseits das Suchen von Kandidaten-Splitpunkten erleichtern und andererseits die durch die Splits entstandenen Partitionen in Bezug auf ihre Reinheit verbessern. Dabei bedient man sich der **Veiteilungsinformationen zu den einzelnen Clustern**. Die Verteilungsinformation zu den Ausprägungen einer gesamten Dimension ist nur bedingt nutzbar. Das Potential von Hilfsinformationen zu den Dimensionen im Vergleich zu Hilfsinformationen zu den Clustern soll anhand eines kurzen Beispiels in Abbildung 26 veranschaulicht werden.

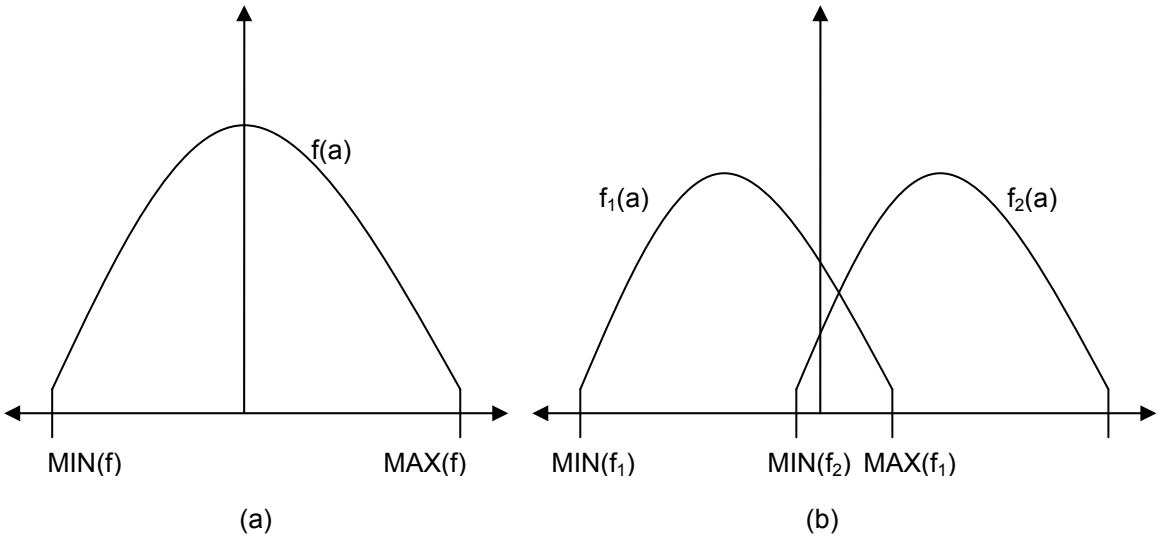


Abbildung 26: Vergleich von Hilfsinformationen zu Dimensionen und zu Clustern

In Abbildung 26 (a) ist die Verteilungskurve f(a) für ein Attribut a dargestellt. Des weitern sind das Minimum MIN(f) und das Maximum MAX(f) dieser Verteilung als Hilfsinformation bekannt. MIN(f) und MAX(f) begrenzen die Attributausprägungen von a, es ist jedoch nicht möglich, Aussagen über die Verteilung der Attributwerte innerhalb der Cluster zu treffen. In Abbildung 26 (b) sind für das gleiche a die Verteilungen der Ausprägungen innerhalb der Cluster veranschaulicht. Das Wissen über die Verteilung ist durch die Minima und Maxima der einzelnen Cluster gegeben. Dadurch ist es möglich, die Daten verschiedener Cluster zu unterscheiden und Splitpunkte in trivialer (MIN(f_1), MIN(f_2), MAX(f_1), MAX(f_2)) oder berechneter Form (Schnittpunkt von f_1 und f_2) zu identifizieren.

Aus diesen Überlegungen heraus lassen sich aus allen identifizierten Hilfsinformationen jene herausfiltern, die Einfluss auf die Qualität des Klassifikators nehmen.
Die als Kennzahlen identifizierten Hilfsinformationen zu den Dimensionen sind bei der Entscheidungsbaum-Konstruktion von sehr geringer Bedeutung, da sie keine Aussagen über die Verteilung der Trainingsdaten in ihre dazugehörigen Klassen treffen. Die Verwendung dieser Informationen bei der Entwicklung einer Splitstrategie ist somit nicht sinnvoll. Daher sind sie a priori von der weiteren Untersuchung auszuschließen.

Unter den Hilfsinformationen zu den Clustern können nur solche Informationen Einfluss auf die Qualität des Klassifikators nehmen, die bei der Durchführung des Split-Prozesses Verwendung finden. Die Werte „Summe“, „Quadratsumme“, „Anzahl der Punkte“ und „Wurzel über die Anzahl der Punkte“ dienen teilweise zur Berechnung weiterer (nützlicher) Hilfsinformationen, besitzen jedoch selbst kein Potential hinsichtlich eines Qualitätseinflusses auf den Klassifikator.
Alle Hilfsinformationen, die Einfluss auf die Qualität des Klassifikators nehmen, gelten hier als **„potentiell verwertbar“**. Sie werden nachfolgend genauer erläutert:

- **Hilfsinformationen – Lageeigenschaften der Trainingsdaten**
  Die Trainingsdaten sind durch das Clusteringergebnis vorgegeben. Sie sind einerseits als zufällige Stichprobe aus dem Clusteringergebnis vorhanden, andererseits in Form von „repräsentativen Daten“ (siehe Abschnitt 6.1) in eigenständigen Tabellen abgelegt. Es soll überprüft werden, welcher Typ von Trainingsdaten zu einem Klassifikator mit der geringsten Fehlerrate führt. Je nach Lage der Trainingsdaten in den Clustern wird der Entscheidungsbaum-Klassifikator unterschiedlich tolerant bzw. genau bezüglich der zu klassifizierenden Testdaten sein.
  Bei der Verwendung von den Clustermittelpunkten (Centroide) sehr **nahen Punkten**, ist die Wahrscheinlichkeit sehr groß, dass Daten, die sich im Grenzbereich von Clustern befinden, falsch klassifiziert werden. Es besteht außerdem eine geringe Wahrscheinlichkeit, dass sich Ausreißer und Fehlerdaten (vgl. [ES00]) in den Trainingsdaten befinden. Der Radius, der durch die nahen Punkte aufgespannten Cluster, ist kleiner als bei der Verwendung von Zufallsdaten, d.h. die Cluster liegen weiter von einander entfernt und es existieren größere „leere“ Bereiche zwischen den Clustern. Die Cluster lassen sich einfacher von einander trennen und beinhalten Punkte, die den Cluster in allen Dimensionen besonders genau repräsentieren.
  Dagegen ist es bei der Verwendung von den Centroiden weit **entfernten Punkten** sehr wahrscheinlich, dass Ausreißer und Fehlerdaten in der Trainingsmenge enthalten sind. Neue Daten, die sich im Grenzbereich von Clustern befinden, werden größerer Sicherheit richtig klassifiziert, da die Trainingsdaten weit außen liegende Repräsentanten enthalten. Die Cluster selbst sind von ihrer Größe her unverändert, d.h. der Radius ist gleich groß

wie bei der Verwendung von Zufallsdaten. Die entfernten Punkte begrenzen die Cluster nach außen hin. Die Abbildungen 27 (a) und (b) zeigen die Trainingsdaten bestehend aus nahen und entfernten Punkten.

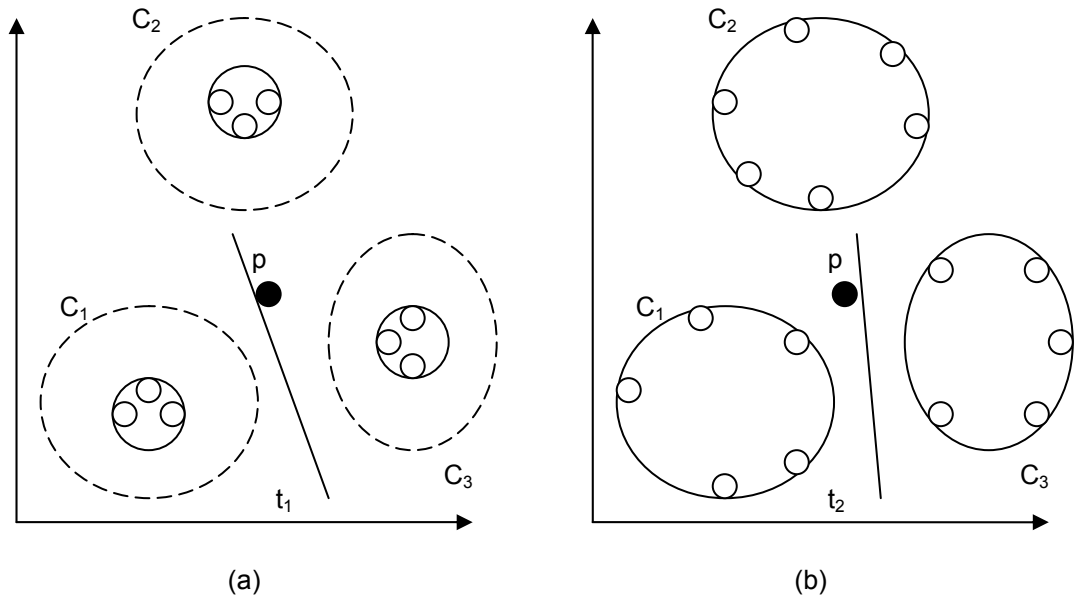


Abbildung 27: Cluster mit nahen und entfernten Punkten

Abbildung 27 (a) zeigt die Cluster der zufälligen Ausgangsdaten mit dazugehörigen Clustern aus nahen Punkten. Die durch die nahen Punkte entstehenden Trenngerade t_1 zwischen C_1 und C_3 würde den neuen Punkt p dem Cluster C_3 zuordnen. In Abbildung 27 (b) werden entfernte Punkte zur Repräsentation der Klassen verwendet. Punkt p wir Cluster C_1 zugeordnet, da die Trenngerade t_2 die Cluster anhand der entfernten Punkte trennt.

Welcher Typ von Trainingsdaten welchen Einfluss auf die Qualität und Güte des Entscheidungsbaum-Klassifikator hat, wird später in dieser Arbeit durch die Testergebnisse bestimmt.

- **Hilfsinformationen – Verteilungsinformationen zu den Clustern**
  Die identifizierten Hilfsinformationen, die Verteilungsinformation zu den Clustern beinhalten, sind Minima der Cluster, Maxima der Cluster, Standardabweichungen der Cluster und Mittelwerte der Cluster (vgl. [SK04]).

Durch die Kombination von Minimum und Maximum bzw. Mittelwert und Standardabweichung ist es möglich, neue Splitstrategien für die Partitionierung von numerischen Attributen zu entwickeln. Diese Strategien werden im Folgenden als „Min/Max-Strategie“ und „Dichtefunktion-Strategie“ bezeichnet.

- Min/Max-Strategie

  Bei dieser Splitstrategie werden die Minima und Maxima der einzelnen Cluster der Trainingsdaten als Splitpunktkandidaten herangezogen. Jener Splitpunktkandidat, der den größten Informationsgewinn liefert, wird als optimaler Split in einer Dimension identifiziert.

  Darüber hinaus ist es möglich, die Überlappungen der Cluster in einer Dimension zu bestimmen. Jener Cluster, der in seinem Wertebereich die übrigen Cluster am geringsten überlappt, wird abgetrennt, d.h. der Split wird entweder am Minimum oder am Maximum dieses Clusters („minOverlapCluster“) durchgeführt. Durch diese Vorgehensweise werden möglichst früh Blattknoten erzeugt, sodass die Klasse für einen Teil der Testdaten sehr schnell festgestellt werden kann, die Eruierung der Klassen der übrigen Testdaten jedoch äußerst langwierig ist. Problematisch an dieser Strategie ist, dass im schlimmsten Fall immer nur ein Cluster von den übrigen getrennt wird. Das kann einen stark negativen Einfluss auf die Höhe des Entscheidungsbaumes haben, da der Baum sehr unbalanciert aufgebaut wird. Das kann genau dann passieren, wenn sich mehrere Cluster nicht miteinander überschneiden. Daher wird von dieser Vorgehensweise abgesehen. Abbildung 28 zeigt den „optimalen Splitpunkt“ bei der Verwendung von minOverlapCluster und einen besseren Splitpunkt bei der Verwendung von Minimum und Maximum als Splitpunktkandidaten.

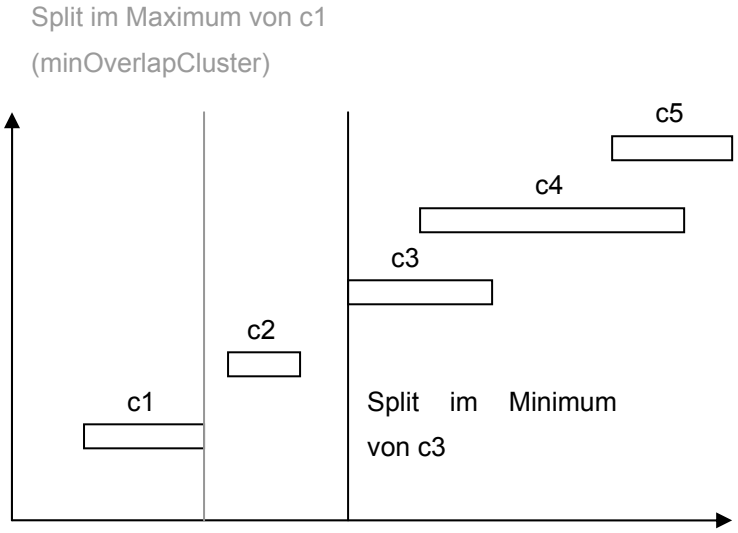


Abbildung 28: Split mit minOverlapCluster

Die Cluster c1 und c2 überlappen die anderen Cluster nicht. Bei einem Split im Maximum von minOverlapCluster c1 wird ein Cluster vollkommen abgetrennt. Die übrige Partition enthält jedoch noch Datensätze von vier Clustern und muss daher noch mehr als ein Mal in anderen Dimensionen gesplittet werden. Dies erhöht die Höhe des Baumes um eine Ebene pro Split im Vergleich zum Split im Minimum von c3. Der Split bei c3 erzeugt zwei Partitionen mit zwei und drei Klassen.

Aus diesem Grund werden bei der Implementierung der Splitstrategie lediglich die einzelnen Minima und Maxima als Splitpunktkandidaten verwendet.

- Dichtefunktion-Strategie

  Für die Dichtefunkton-Strategie werden die Hilfsinformationen Mittelwert und Standardabweichung benötigt. Die Idee dieser Strategie liegt in der Berechnung der Schnittpunkte der Dichtefunktionen einzelner Cluster. Jeder Schnittpunkt wird dann als Splitpunktkandidat angenommen. Der Schnittpunkt mit dem größten Informationsgewinn stellt den optimalen Split in der jeweiligen Dimension dar.

Unter der Annahme, dass es sich bei den Trainingsdaten T um eine normal verteilte Menge von Punkten N(μ, σ) der Dimensionen a_1, …, a_m handelt, ist die Dichtefunktion einer Verteilung wie folgt definiert [BH03]:

$$\phi(z) = f(z) = \frac{1}{2 \cdot \sqrt{\pi}} \cdot e^{-\frac{1}{2}(z)^2} \quad \text{für} \quad z = \frac{x - \mu}{\sigma}, \text{ wobei } x \in T, \text{ μ der Mittelwert}$$

von $a_m \in T$ und σ die Standardabweichung von $a_m \in T$ sind.

Der Schnittpunkt zweier Dichtefunktionen $f(z_1) = f(z_2)$ befindet sich an der Stelle, an der die Werte für z_1 und z_2 identisch sind. Zwei Dichtefunktionen können jedoch bis zu vier Schnittpunkte besitzen. Liegen die Dichtefunktionen wie in Abbildung 29, so schneiden sie sich in den Punkten x_1 und x_2. Da sich die Funktionswerte logarithmisch gen Null bewegen, existieren noch zwei weitere Schnittpunkte, die in Abbildung 26 als x_3 und x_4 angedeutet sind.

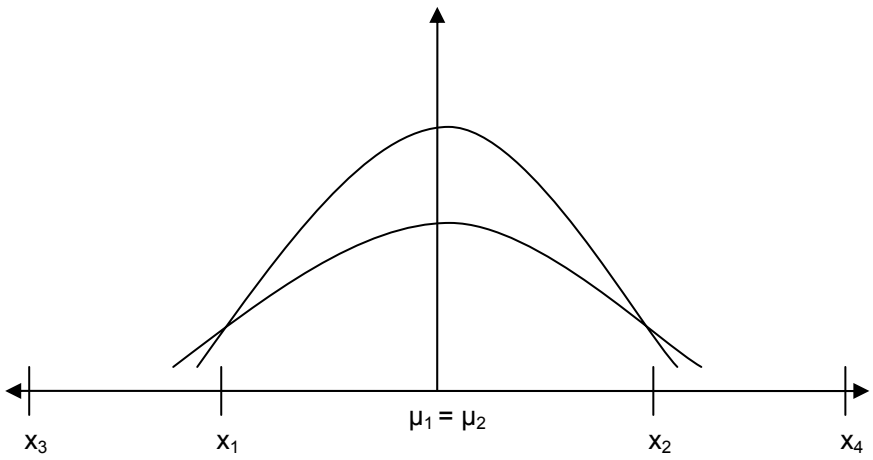


Abbildung 29: Dichtefunktionen mit vier Schnittpunkten

In diesem Fall liegen die Ausprägungen der beiden Verteilungen eng beieinander oder sind identisch mit unterschiedlichen Häufigkeiten. Der Versuch eines Splits bei dieser Wertekonstellation führt zu sehr unreinen Partitionen und wird normalerweise nicht durchgeführt.

Im Normalfall sind nur jene Schnittpunkte als Splitpunktkandidaten relevant, die zwischen den Mittelwerten der der Verteilungen liegen und

die Verteilungen somit von einander trennen. Abbildung 30 zeigt die Dichtefunktionen zweier Cluster mit einem relevanten Schnittpunkt x_1.

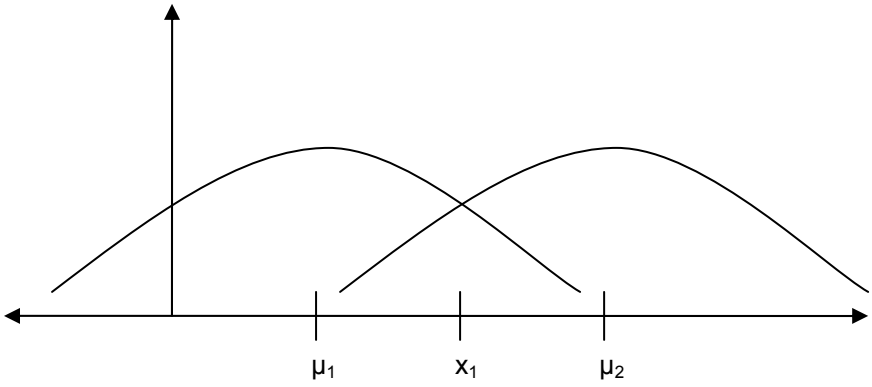


Abbildung 30: Relevanter Schnittpunkt zweier Dichtefunktionen

Setzt man z_1 und z_2 gleich, um den Schnittpunkt zweier Dichtefunktionen zu berechnen, so erhält man folgende Gleichung:

$$\left|\frac{x-\mu_1}{\sigma_1}\right| = \left|\frac{x-\mu_2}{\sigma_2}\right|,$$ wobei $\mu_{1,2}$ und $\sigma_{1,2}$ Mittelwerte und Standardabweichungen der zwei Cluster darstellen, deren Werte die Dichtefunktionen aufspannen. Je nachdem, ob die Mittelwerte und Standardabweichungen positiv oder negativ sind, erhält man durch Auflösung des Betrages einen bestimmten Schnittpunkt entlang der x-Achse.

Formt man nach Auflösung des Betrages die Gleichung nach x um, so erhält man im Normalfall den Wert x', in dem sich die Dichtefunktionen der Cluster schneiden:

$$x' = \frac{\sigma_2 \cdot \mu_1 + \sigma_1 \cdot \mu_2}{\sigma_1 - \sigma_2}$$

Befindet sich x' zwischen den Mittelwerten der Cluster, so wird x' als Splitpunktkandidat angenommen. Ist dies nicht der Fall so wird der Schnittpunkt x'' berechnet. Bei diesem Schnittpunkt geht man von folgendern Schnittgleichungen aus:

$$\frac{x-\mu_1}{\sigma_1}=\frac{\mu_2-x}{\sigma_2} \text{ und } \frac{\mu_1-x}{\sigma_1}=\frac{x-\mu_2}{\sigma_2}$$

In beiden Fällen erreicht man, nach einer Umformung der Gleichungen nach x, einen relevanten Schnittpunkt x'', der zwischen den Mittelwerten der beiden geschnittenen Dichtefunktionen liegt:

$$x''=\frac{\sigma_2\cdot\mu_1+\sigma_1\cdot\mu_2}{\sigma_1+\sigma_2}$$

Je nach Lage der Schnittpunkte wird für x' oder x'' der Informationsgewinn berechnet. Dieser wird dann mit den Informationsgewinnwerten der übrigen Splitpunktkandidaten verglichen. Der Splitpunktkandidat mit dem höchsten Informationsgewinn wird als bester Split in der Dimension angenommen.

Die im folgenden Kapitel erläuterte Implementierung zur Konstruktion der Entscheidungsbäume berücksichtigt nur Hilfsinformationen, die hier als „potentiell verwertbar“ erachtet werden.

7 Implementierung

Nachdem die Grundprinzipien des „Kombinierten Data Minings“ erläutert wurden, zeigt dieses Kapitel die Realisierung des Typus „Nachfolgerverfahren kennt Vorgängerverfahren“ durch die Implementierung eines Entscheidungsbaum-Klassifikators, der auf das RainForest-Framework basiert. Der implementierte Algorithmus verwendet Hilfsinformationen, die im Vorgängerverfahren „K-Means“ identifiziert wurden [SK04], um die Brauchbarkeit dieser Hilfsinformationen in Bezug auf die Klassifikatorqualität zu bestimmen.
Nachfolgend wird die Implementierung des Entscheidungsbaum-Klassifikators detailliert erläutert. Programmteile, in denen Hilfsinformationen verwendet werden, werden im Besonderen anhand von Quellcodeauszügen betrachtet. Darüber hinaus werden Codepassagen zur Erstellung des Baumes in der Datenbank, zur Evaluierung der Klassen der Testdaten und zur Bestimmung der Qualitätskriterien genauer beleuchtet.

7.1 Architektur

Ehe auf die Implementierung eingegangen werden kann, muss zunächst die Architektur vorgestellt, die bei der Umsetzung des Entscheidungsbaum-Klassifikators zum Einsatz gekommen ist.
Die Trainings- und Testdaten sowie die Hilfsinformationen stammen aus einer Datenbank, wo sie vom Vorgängerverfahren gespeichert wurden. Die Implementierung erfolgte in C++. Als Entwicklungsumgebung wurde „Microsoft Visual Studio 6.0“ verwendet. Der Datenbankzugriff wurde mittels „Embedded SQL“ (E/SQL) umgesetzt. Um den E/SQL-Code in eine C-konforme Repräsentation umzuwandeln, wurde ProC/C++ als Precompiler eingesetzt. Abbildung 31 stellt die Architektur der Implementierung grafisch dar.

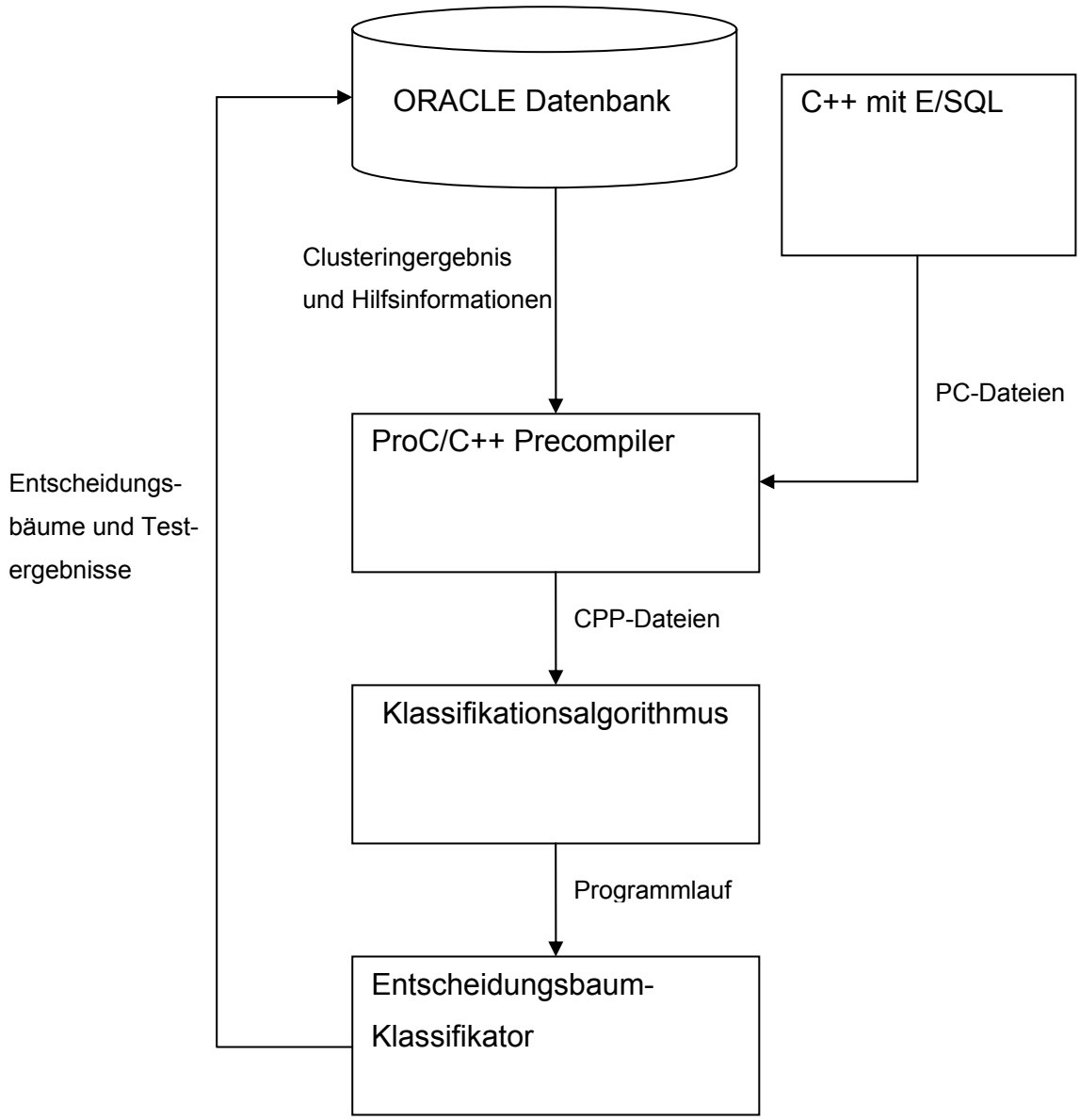


Abbildung 31: Architektur

In den PC-Dateien werden C++-Code und E/SQL-Code gemischt implementiert. Der ProC/C++ Precompiler generiert aus dem E/SQL-Code C-konforme Konstrukte, die in weitere Folge als CPP-Dateien kompiliert werden können. Die Programmausführung erzeugt (unter der Zuhilfenahme von Hilfsinformationen) einen Entscheidungsbaum-Klassifikator, dessen Qualität anhand von Testdaten bestimmt wird. Die erstellten Entscheidungsbäume werden gemeinsam mit den klassifizierten Testdaten zurück in die Datenbank geschrieben.

7.2 Dokumentation des Quellcodes – DT-Klassifikator

Zur Implementierung des hier verwendeten Decision-Tree-Klassifikators wurde das Framework „RainForest" [RGG98] herangezogen. Das Framework dient jedoch nur zum Teil als Grundlage für den entwickelten Source-Code. Zwar werden sehr wohl AVC-Gruppen (siehe Kapitel 5.4) zur Evaluierung des besten Splits verwendet, es ist jedoch nicht möglich verschiedene, austauschbare Algorithmen über die Schnittstelle der Implementierung zu übergeben. Die Reinheit der durch die Splitkandidaten erzeugten Partitionen wird ausschließlich durch den Informationsgewinn bestimmt.
Der entwickelte Algorithmus, im Folgenden auch **DT-Algorithmus** genannt, besitzt mehrere festgeschriebene Splitstrategien (siehe Kapitel 6.2). Der Typ der zu durchlaufenden Splitstrategie wird während der Ausführung des Algorithmus als Parameter übergeben und während der Klassifikator-Generierung berücksichtigt. Es wird außerdem davon ausgegangen, dass die gesamte AVC-Gruppe der Baumwurzel in den Hauptspeicher passt (vgl. RF_Write in Kapitel 5.4).
Das UML-Diagramm in Abbildung 29 zeigt die Klassentopologie der Implementierung. Die Datenbankzugriffe werden in Form von E/SQL-Code an den entsprechenden Stellen eingefügt. Es existiert keine explizite Klasse für den Datenbankzugriff. Jede Klasse enthält zusätzlich zu den in Abbildung 29 dargestellten Methoden einen Standardkonstruktor und einen Destruktor.

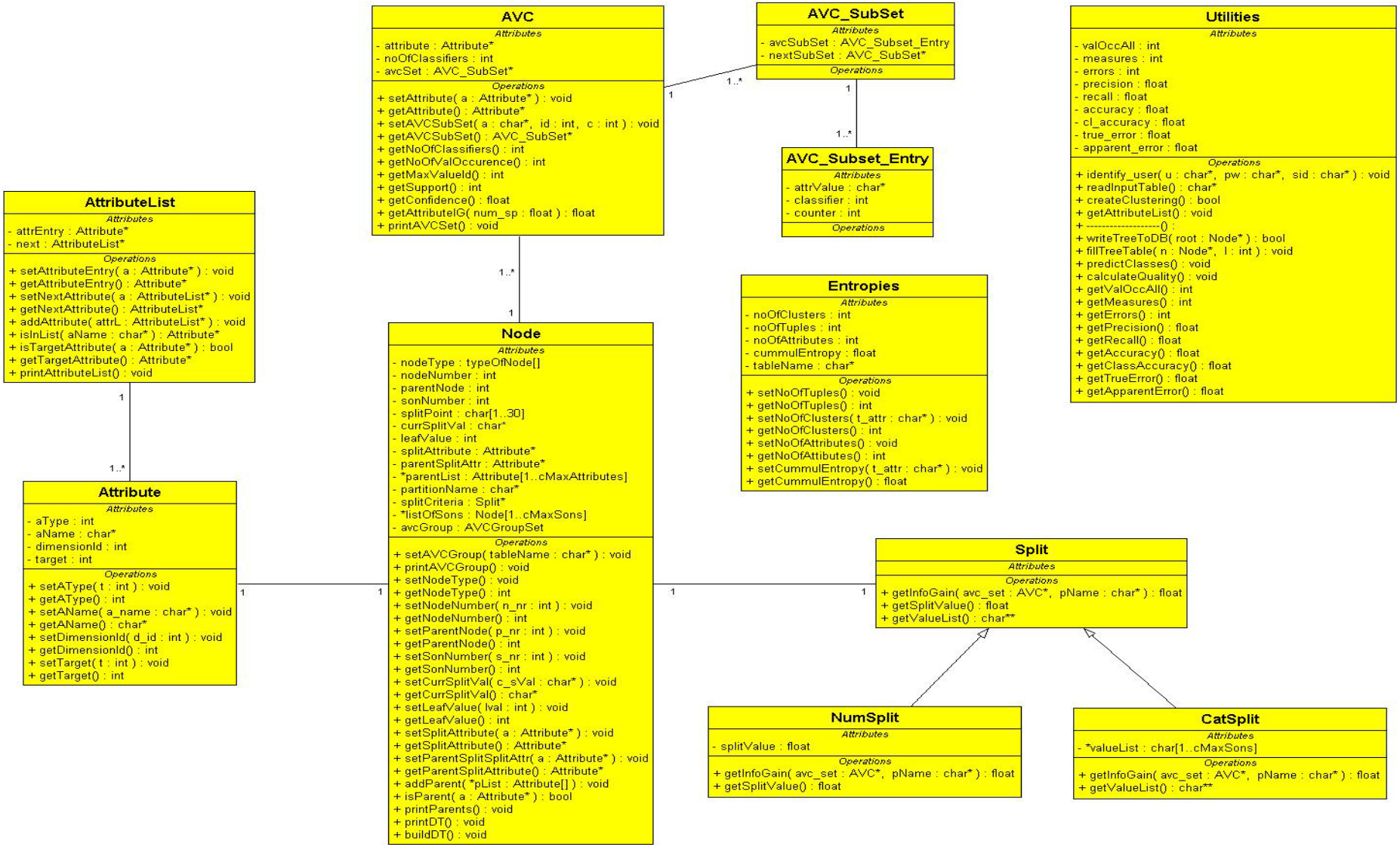


Abbildung 32: UML-Diagramm des DT-Algorithmus

Es folgt eine Erläuterung der Klassen und ihrer wichtigsten Attribute und Methoden:

- **Utilities**

  Die Klasse Utilities enthält Methoden und Attribute für die Initialisierung der Trainings- und Testdaten. Sie enthält außerdem alle Methoden zur Durchführung der Testläufe, zur Erstellung der Entscheidungsbäume in der Datenbank und zur Berechnung der Qualitätskriterien der Entscheidungsbäume. Die Klasse enthält folgende Attribute und Methoden:

Attribute	
valOccAll	Anzahl der Testtupel
measures	Richtig klassifizierte Testtupel
errors	Falsch klassifizierte Testtupel
precision	Anteil der Tupel, die richtig klassifiziert wurden unabhängig davon, ob sie der zugeordneten Klasse angehören
recall	Anteil der Tupel, die der zugeordneten Klasse angehören und richtig klassifiziert wurden
accuracy	Klassifikationsgenauigkeit bzgl. der Relevanz
cl_accuracy	Klassifikationsgenauigkeit bzgl. der Testdaten
true_error	Tatsächlicher Klassifikationsfehler
apparent_error	Beobachteter Klassifikationsfehler
Methoden	
Identify_user(char* u, char* pw, char* sid)	Methode zum Einlesen von User, Passwort und Datenbank-ID
readInputTable()	Methode zum Einlesen der Input-Tabelle; gibt das Ziel-Attribut zurück
createClustering()	Methode zur Erzeugung der Trainingstabelle aus der der DT erzeugt werden soll; gibt TRUE bei erfolgreicher Erstellung der Tabelle zurück, sonst

	FALSE
getAttributeList(char* t_attr)	Methode zur Erstellung eine Liste der Attribute der Trainingsdaten;
writeTreeToDB(Node* root)	Methode, die den generierten DT in eine ORACLE-Tabelle schreibt; gibt TRUE bei erfolgreicher Generierung der Baum-Tabelle zurück, sonst FALSE
fillTreeTable(Node* n, int l)	Rekursive Methode zum Füllen der Baum-Tabelle
predictClasses()	Methode, die mithilfe der Baum-Tabelle die voraussichtlichen Klassen der Testtupel bestimmt und diese in eine Tabelle resultTable schreibt
calculateQuality()	Methode zur Berechnung der Qualitätskriterien (precision, recall, accuracy, cl_accuracy, true_error und apparent_error)
getValOccAll()	Methode zur Rückgabe der Anzahl der Testtupel (valOccAll)
getMeasures()	Methode zur Rückgabe der Anzahl der richtig klassifizierten Testtupel (measures)
getErrors()	Methode zur Rückgabe der Anzahl der falsch klassifizierten Testtupel (errors)
getPrecision()	Methode zur Rückgabe der Anzahl der richtig klassifizierten Tupel, unabhängig von der zugehörigen Klasse (precision)
getRecall()	Methode zur Rückgabe der Anzahl der richtig klassifizierten Tupel bzgl. ihrer Relevanz (recall)
getAccuracy()	Methode zur Rückgabe der Klassifikationsgenauigkeit bzgl. der Relevanz der Testtupel (accuracy)
getClassAccuracy()	Methode zur Rückgabe der

	Klassifikationsgenauigkeit, unabhängig von der Relevanz (cl_accuracy)
getTrueError()	Methode zur Rückgabe des tatsächlichen Klassifikationsfehlers (true_error)
getApparentError()	Methode zur Rückgabe des beobachteten Klassifikationsfehlers (apparent_error); gibt bei Test über die Trainingsdaten apparent_error zurück, sonst -1

Tabelle 12: Attribute und Methoden der Klasse „Utilities“

- **Entropies**

  Die Klasse „Entropies“ beinhaltet Attribute und Methoden zur Berechnung von Metadaten zu den Trainingsdaten. Darüber hinaus enthält sie die Methode zur Berechnung der Gesamt-Entropie der Trainingsdaten. Folgende Attribute und Methoden sind in „Entropies“ enthalten:

Attribute	
noOfClusters	Anzahl der Cluster der Trainingsdaten
noOfTuples	Anzahl der Tupel der Trainingsdaten
noOfAttributes	Anzahl der Attribute der Trainingsdaten
cummulEntropy	Gesamt-Entropie der Trainingsdaten
tableName	Name der Trainingstabelle
Methoden	
setNoOfClusters(char* t_attr)	Methode zur Berechnung der Anzahl der Cluster der Trainingstabelle (noOfClusters)
getNoOfClusters()	Methode zur Rückgabe der Anzahl der Cluster der Trainingstabelle (noOfClusters)
setNoOfTuples()	Methode zur Berechnung der Anzahl der Tupel der Trainingstabelle (noOfTuples)

getNoOfTuples()	Methode zur Rückgabe der Anzahl der Tupel der Trainingstabelle (noOfTuples)
setNoOfAttributes()	Methode zur Berechnung der Anzahl der Attribute der Trainingstabelle (noOfAttributes)
getNoOfAttributes()	Methode zur Rückgabe der Anzahl der Attribute der Trainingstabelle (noOfAttributes)
setCummulEntropy(char* t_attr)	Methode zur Berechnung der Gesamt-Entropie der Trainingstabelle (cummulEntropy)
getCummulEntropy()	Methode zur Rückgabe der Gesamt-Entropie (cummulEntropy)

Tabelle 13: Attribute und Methoden der Klasse „Entropies“

- **Attribute**

  Die Klasse „Attribute“ enthält Attribute und Methoden zu den Attributen der Trainingstabelle. Instanzen dieser Klasse sind in einer zur Laufzeit verwendeten Attributliste vorhanden. Nachfolgend wird ihr Inhalt tabellarisch erläutert:

Attribute	
aType	Typ des Attributs (0 für numerisches, 1 für kategorisches und -1 für undefiniertes Attribut)
aName	Name des Attributs
dimensionId	Dimensions-Id bzgl. der Hilfsinformationstabellen (1..10)
target	Ziel-Attribut-Indikator (0 für kein Ziel-Attribut, 1 für Ziel-Attribut
Methoden	
setAType(int t)	Methode zum Setzen des Typs eines Attributs (aType)

getAType()	Methode zur Rückgabe des Typs eines Attributs (aType)
setAName(char* a_name)	Methode zum Setzen des Namens eines Attributs (aName)
getAName()	Methode zur Rückgabe des Namens eines Attributs (aName)

Tabelle 14: Attribute und Methoden der Klasse „Attribute“

- **AttributeList**

  Die Klasse „AttributeList“ beinhaltet Attribute und Methoden zur Erstellung und Verwendung einer Attributliste. Sie enthält Objekte der Klasse Attribute und einen Zeiger (pointer) auf das nächste Objekt in der Attributliste. Die Attribute und Methoden der Klasse AttributeList sind folgende:

Attribute	
attrEntry	Attribut-Eintrag in der Attributliste
next	Zeiger auf den nächsten Attributlisten-Eintrag
Methoden	
setAttributeEntry(Attribute* a)	Methode zum Setzen der Attributwerte des Attributlisten-Eintrags
getAttributeEntry()	Methode zur Rückgabe des aktuellen Attributs der Attributliste
setNextAttribute(AttributeList* a)	Methode zum Setzen des Zeigers auf den nächsten Attributlisten-Eintrag
getNextAttribute()	Methode zur Rückgabe einer Referenz auf den nächsten Attributlisten-Eintrag
addAttribute(AttributeList* attrL)	Methode zum Hinzufügen eines neuen Eintrages (Attribut) in die Attributliste; das Attribut wird hinten an die Liste angefügt
isInList(char* aName)	Methode, die prüft, ob ein Attribut mit übergebenem Namen in der Attributliste

	enthalten ist; gibt eine Referenz auf das Attribut zurück, wenn ein Attribut mit Namen aName existiert, sonst NULL
isTargetAttribute(Attribute* a)	Methode, die prüft, ob ein Attribut ein Ziel-Attribut ist; gibt TRUE zurück, wenn das übergebene Attribut Ziel-Attribut ist, sonst FALSE
getTargetAttribute()	Methode zur Rückgabe des Ziel-Attributs der Attributliste
printAttributeList()	Methode zur Ausgabe der Attributliste auf dem Bildschirm

Tabelle 15: Attribute und Methoden der Klasse „AttributeList“

- **AVC**

  Die Klasse „AVC“ enthält als Attribute und Methoden die Datenstruktur der AVC-Mengen und deren Zugriffsroutinen. Für jede AVC-Menge eines Knotens existiert eine Instanz der Klasse „AVC“. Die Attribute und Methoden der Klasse gliedern sich wie folgt:

Attribute	
attribute	Name des Attributs, zu dem die AVC-Menge gehört
noOfClassifiers	Anzahl der unterschiedlichen Klassen des Ziel-Attributs im AVC-Set
avcSet	Kopf (head) der AVC-Menge
Methoden	
setAttribute(Attribute* a)	Methode zum Setzen des Attributs der AVC-Menge (attribute)
getAttribute()	Methode zur Rückgabe des Attributs der AVC-Menge (attribute)
setAVCSubSet(char* a, int id, int c)	Methode zum Setzen des Wertes eines AVC-SubSets, d.h. die Kombination einer Attributausprägung mit einer Klasse im AVC-Set

getAVCSubSet()	Methode zur Rückgabe eines AVC-SubSets einer AVC-Menge
getNoOfClassifiers()	Methode, die die Anzahl der unterschiedlichen Klassen im AVC-Set berechnet und zurückgibt
getNoOfValOccurrence()	Methode, die die Anzahl der unterschiedlichen Attributausprägungen im AVC-Set berechnet und zurückgibt
getMaxValueId()	Methode, die jene Klassen-Id berechnet, deren Klasse die meisten Tupel der Knoten-Partition angehören
getSupport()	Methode, die die Anzahl der zu einem AVC-Set gehörigen Tupel berechnet und zurückgibt
getConfidence()	Methode, die die Konfidenz jener Klasse berechnet und zurückgibt, der die meisten Tupel der Knoten-Partition angehören
getAttributeIG(float num_sp)	Methode, die den Informationsgewinn aus der AVC-Menge eines Attributs berechnet und zurückgibt
printAVCSet()	Methode, die ein AVC-Set am Bildschirm ausgibt

Tabelle 16: Attribute und Methoden der Klasse „AVC“

- **Split**

  Die Klasse „Split“ ist eine abstrakte Klasse und enthält die Eigenschaften einer Splitkategorie (numerisch oder kategorisch) eines Knotens. Sie ist Superklasse der beiden Subklassen „NumSplit“ und „CatSplit“. Da die Klasse „Split“ keine Attribute enthält und keine ihrer Methoden implementiert werden sie hier nicht angeführt. Die Subklassen implementieren alle in „Split“ deklarierten Methoden.

- **NumSplit**

  Die Klasse „NumSplit“ enthält Attribute und Methoden zum Vergleichswert eines numerischen Splitpunktes. Sie ist von der Klasse „Split“ abgeleitet und implementiert einen Teil ihrer Methoden. Ihre Attribute und Methoden lauten wie folgt:

Attribute	
splitValue	Wert des numerischen Splitpunktes
Methoden	
getInfoGain(AVC* avc_set, char* pName)	Methode, die alle möglichen Splitpunkte eines numerischen Splits hinsichtlich ihres Informationsgewinns evaluiert und den besten Splitpunkt (splitValue) berechnet bzw. setzt
getSplitValue()	Methode zur Rückgabe des numerischen Splitpunktes (splitValue)

Tabelle 17: Attribute und Methoden der Klasse „NumSplit“

- **CatSplit**

  Die Klasse „CatSplit“ enthält Attribute und Methoden zu den Vergleichswerten eines kategorischen Splits. Sie ist von der Klasse „Split“ abgeleitet und implementiert einen Teil ihrer Methoden. Sie enthält folgende Attribute und Methoden:

Attribute	
valueList	Liste mit alle Ausprägungen (Vergleichswerten) eines kategorischen Split-Attributs
Methoden	
getInfoGain(AVC* avc_set, char* pName)	Methode zur Berechnung des Informationsgewinns beim Split über ein kategorisches Attribut

getValueList()	Methode zur Rückgabe der Vergleichswerte eines kategorischen Split-Attributs

Tabelle 18: Attribute und Methoden der Klasse „CatSplit“

- **Node**

  Die Klasse „Node“ bildet das Kernstück der Implementierung des DT-Algorithmus. Sie enthält Attribute und Methoden zur Erstellung der Knoten des Entscheidungsbaumes. Jede Instanz von „Node“ beinhaltet eine AVC-Gruppe und das Splitkriterium des aktuellen Knoten. Die Klasse enthält darüber hinaus eine rekursive Methode zur Generierung des gesamten Entscheidungsbaumes. Nachfolgend sind ihre Attribute und Methoden beschrieben:

Attribute	
nodeType	Typ des Knotens (root für Wurzelknoten, intern für inneren Knoten und leaf für Blattknoten); nodeType ist vom Datentyp „enum“ und wird Programmintern als INTEGER repräsentiert (0 für root, 1 für intern, 2 für leaf)
nodeNumber	Laufende Knotennummer (root = 0)
parentNumber	Laufende Nummer des Vaterknotens
sonNumber	Sohnnummer des aktuellen Knotens (0 und 1 bei numerischem Split, 0 bis 5 bei kategorischem Split)
splitPoint	Splitpunkte des Knotens
currSplitVal	Splitpunkt als berechneter Splitwert (numerisches Attribut) oder Attributausprägung (kategorisches Attribut)
leafValue	Klasse des aktuellen (Blatt-)Knotens (0 für Nicht-Blattknoten, Klasse 1..6 für Blattknoten)

splitAttribute	Split-Attribut
parentSplitAttr	Split-Attribut des Vaterknotens
parentList	Liste mit Referenzen aller Vaterknoten des aktuellen Knotens
partitionName	Name der Tabellenpartition des Knotens, aus der die AVC-Gruppe erzeugt wird
splitCriteria	Enthält alle möglichen Vergleichswerte für einen Split über ein numerisches oder kategorisches Split-Attribut (splitAttribute); splitCriteria ist ein Objekt der Klasse Split und verfügt über Methoden zur Berechnung des Informationsgewinnes für einen bestimmten Splitpunkt
listOfSons	Liste mit Zeigern auf alle Sohnknoten des aktuellen Knotens
avcGroup	AVC-Gruppe des Knotens
Methoden	
setNodeType()	Methode zum Setzen des Knotentyps (nodeType)
getNodeType()	Methode zur Rückgabe des Knotentyps (nodeType)
setNodeNumber(int n_nr)	Methode zum Setzen der Knotennummer (nodeNumber)
getNodeNumber()	Methode zur Rückgabe der Knotennummer (nodeNumber)
setParentNode(int p_nr)	Methode zum Setzen der Knotennummer des Vaterknotens (parentNode)
getParentNode()	Methode zur Rückgabe der Knotennummer des Vaterknotens (parentNode)
setSonNumber(int s_nr)	Methode zum Setzen der Sohnnummer des aktuellen Knotens (sonNumber)

getSonNumber()	Methode zur Rückgabe der Sohnnummer des aktuellen Knotens (sonNumber)
setAVCGroup(char* tableName)	Methode zur Erstellung der AVC-Gruppe des aktuellen Knotens aus der übergebenen Tabellenpartition (= Trainingsmenge beim Wurzelknoten)
setCurrSplitVal(char* c_sVal)	Methode zum Setzen des Wertes des Splitpunktes am Knoten (currSplitVal)
getCurrSplitVal()	Methode zur Rückgabe des Wertes des Splitpunktes am Knoten (currSplitVal)
setLeafValue(int l_val)	Methode zum Setzen des Blattknotenwertes (leafValue)
getLeafValue()	Methode zur Rückgabe des Blattknotenwertes (leafValue)
setSplitAttribute(Attribute* a)	Methode zum Setzen des Split-Attributs (splitAttribute)
getSplitAttribute()	Methode zur Rückgabe des Split-Attributs (splitAttribute)
setParentSplitAttr(Attribute* a)	Methode zum Setzen des Split-Attributs des Vaterknotens (parentSplitAttr)
getParentSplitAttr()	Methode zur Rückgabe des Split-Attributs des Vaterknotens (parentSplitAttr)
addParent(Attribute *pList[])	Methode, die der Vaterknoten-Liste eine Referenz auf einen Vaterknoten hinzufügt
isParent(Attribute* a)	Methode, die prüft, ob das übergebene Attribut an einem Vaterknoten des aktuellen Knotens Split-Attribut ist
printParents()	Methode zur Ausgabe der Vaterknoten des aktuellen Knotens am Bildschirm
printDT()	Methode zur Ausgabe des Entscheidungsbaums oder eines seiner

	Unterbäume am Bildschirm
buildDT()	Rekursive Methode zur Generierung des Entscheidungsbaumes ausgehend vom aktuellen Knoten

Tabelle 19: Attribute und Methoden der Klasse „Node“

Des Weiteren enthält die Implementierung des DT-Klassifikators eine Reihe von globalen Konstanten. Sie sind entweder durch die Ausgangsdaten (Clusteringergebnis) vorgegeben oder mit Werten belegt, die für die Entscheidungsbaum-Konstruktion als sinnvoll erachtet wurden. Ihre Wertigkeiten und Bedeutungen werden nachfolgend erläutert.

```
cMinSupport = 50;       // Minimaler Support eines Blatt-Knotens (5% der Trainingsmenge)
cMinConfidence = 0.6;   // Minimale Konfidenz eines Blattknotens (ca. 2/3 einer Partition)
cMaxClusters = 6;       // Maximale Anzahl von Clustern einer Tabelle
cMaxSons = 6;           // Maximale Anzahl der Söhne eines Baum-Knotens
cMaxAttributes = 11;    // Maximale Anzahl von Attributen im Clustering (inkl. Ziel-Attribut)
cMaxTuples = 5000;      // Maximale Anzahl von Tupeln einer Tabelle
cMaxDigits = 7;         // Maximale Anzahl der Ziffern eines Splitwertes
cMaxPartitions = 100;   // Maximale Anzahl der Trainingsdaten-Partitionen
cMinAValue = -10000.00;         // Minimaler Wert für ein numerisches Attribut
cMaxAValue = 10000.00;          // Minimaler Wert für ein numerisches Attribut
```

7.2.1 DT-Algorithmus

Der implementierte DT-Algorithmus ist in seiner Umsetzung an das in Kapitel 5.4 vorgestellte RainForest-Framework angelehnt. Für jeden Knoten des Baumes wird eine AVC-Gruppe erstellt, mithilfe derer der jeweils beste Split am Knoten evaluiert wird. Bei der Initialisierung des Algorithmus wird zunächst ein Knoten als Wurzelknoten erstellt, für den die AVC-Gruppe aus der gesamten Trainingsmenge ermittelt wird. Von diesem Knoten ausgehend wird der Entscheidungsbaum rekursiv aufgebaut. Der Ablauf des DT-Algorithmus ist in Form eines Auszugs des Quellcodes der main-Methode in Abbildung 33 veranschaulicht. Die einzelnen Schritte bei der Erstellung und den Tests des Entscheidungsbaumes sind als grau hinterlegte

Kommentare im Quellcode markiert. Die Verwendung der Hilfsinformationen ist darin nicht ersichtlich und wird im nachfolgenden Unterkapitel genauer beleuchtet.

```
void main(int argc, char *argv[]) {
        /* ---------------------------------------------- */
        // DATENBANKVERBINDUNG
        /* ---------------------------------------------- */
        // Identifikation des Users
        u.identify_user(*&user,*&pass,*&s_id);
        // Zuweisung von User und Passwort
        strcpy((char *)username.arr,(const char*)user);
        username.len = strlen((char *)username.arr);
        strcpy((char *)password.arr,(const char*)pass);
        password.len = strlen((char *)password.arr);
        strcpy((char *)sid.arr,(char*)s_id);
        sid.len = strlen((char *)sid.arr);
        // Datenbankverbindung aufbauen
        EXEC SQL WHENEVER SQLERROR DO sql_error("Connect error:");
        EXEC SQL CONNECT :username IDENTIFIED BY :password USING :sid;
        /* ---------------------------------------------- */
        // INITIALISIERUNGEN
        /* ---------------------------------------------- */
        // Trainings-Tabelle festlegen
(1)     targetAttribute = u.readInputTable();
        // Attributliste erzeugen
        u.getAttributeList(targetAttribute);
        // Setzen der Metadaten der Trainingsdaten
        e.setNoOfTuples();
        e.setNoOfAttributes();
        e.setNoOfClusters(targetAttribute);
        e.setCummulEntropy(targetAttribute);
        /* ---------------------------------------------- */
        // DT KONSTRUIEREN
        /* ---------------------------------------------- */
        // Splitstrategie für numerische Werte auswählen
        // -----------------------------------------
        printf("\nSplitstrategie (a.. ohne Verteilung, b.. Min/Max, c.. Dichtefunktion): ");
        gets(splitParameter);
        // -------------------------------
        // Wurzelknoten (root) erstellen
        // -------------------------------
```

```
            Node *root = new Node;
            // Erzeugen der AVC-Gruppe des Wurzelknotens
            root->setAVCGroup(inputTableName);
            // Rekursiver Aufbau des Entscheidungsbaumes
(2)         root->buildDT();
            // ----------------------------------------
            // Erzeugten Baum in die Datenbank schreiben
            // ----------------------------------------
            done = u.writeTreeToDB(root);
            // -------------------------------
            // TESTMETHODEN
            // -------------------------------
            // Klassen der Testdaten evaluieren
            u.predictClasses();
            // Berechnen der Qualitätsmerkmale
            u.calculateQuality();
            // Ausgabe der Qualitätskriterien
            printf("\nAnzahl der Testtupel: %d",u.getValOccAll());
            printf("\nAnzahl der richtig Klassifizierten: %d",u.getMeasures());
            printf("\nAnzahl der falsch Klassifizierten: %d\n",u.getErrors());
            printf("\nPrecision: %f",u.getPrecision());
            printf("\nRecall: %f",u.getRecall());
            printf("\nAccuracy: %f",u.getAccuracy());
            printf("\nKlassifikationsgenauigkeit: %f",u.getClassAccuracy());
            printf("\nTatsaechlicher Klassifikationsfehler: %f",u.getTrueError());
            printf("\nBeobachteter Klassifikationsfehler: %f",u.getApparentError());
            // DB-Connection schließen
            EXEC SQL COMMIT WORK RELEASE;
}
```

Abbildung 33: Quellcode des DT-Algorithmus

7.2.2 Verwenden der Hilfsinformationen

Die im Vorgängervefahren „K-Means“ (vgl. [SK04]) ermittelten Hilfsinformationen fließen in den DT-Algorithmus zur Laufzeit ein.

Hilfsinformationen zu den Lageeigenschaften der Trainingsdaten sind implizit in der Trainingsmenge enthalten. Als Trainingsmenge dient eine Tabelle mit nahen und entfernten Punkten. In allen anderen Fällen wird eine zufällige Stichprobe aus den

Ausgangsdaten herangezogen. Der Name der Trainingstabelle ist in der globalen Variable „inputTableName“ abgelegt. Sie wird in der Methode „readInputTable“ (vgl. Zeile (1) in Abbildung 33) der Klasse „Utilities“ eingelesen und gesetzt. Die Methode liefert außerdem das Zielattribut (targetAttribute) der Trainingsmenge.

Die Hilfsinformationen mit Verteilungsinformation werden aus einer separaten Hilfsinformationstabelle gelesen. Sie werden bei der Suche nach Splitpunkten dazu verwendet, potentielle Splitpunkte zu identifizieren und hinsichtlich des besten Splits zu evaluieren. Bei den verwendeten Hilfsinformationen handelt es sich um Hilfsinformationen zu den Clustern. Sie werden genau dann aus der Datenbank (Hilfsinformationstabelle) gelesen, wenn für eine bestimmte Dimension am aktuellen Knoten Splitpunkte bestimmt werden müssen. Die Splitpunkte werden während des rekursiven Aufbaus des Entscheidungsbaumes bestimmt (vgl. Zeile (2) in Abbildung 33).
Der DT-Algorithmus verfügt über drei individuelle Splitstrategien, wobei eine davon gänzlich ohne Hilfsinformationen den „optimalen“ Splitpunkt einer Dimension ermittelt. Die Strategien sind die „Splitstrategie ohne Verteilungsinformation“, die „Min/Max-Strategie“ und die „Dichtefunktion-Strategie“.

Splitstrategie ohne Verteilungsinformation

Diese Splitstrategie geht so vor, dass zunächst die numerischen Attribute einer Dimension aufsteigend sortiert werden. Dann wird immer von zwei benachbarten Attributausprägungen das arithmetische Mittel berechnet. Jeder dieser berechneten Werte wird als Splitpunkt herangezogen und mittels des Informationsgewinnes bewertet. Das arithmetische Mittel mit dem höchsten Informationsgewinn wird als bester Splitpunkt angenommen [ES00]. Abbildung 34 zeigt diese Vorgehensweise anhand eines Auszugs aus dem Quellcode. Die Kommentare zu den beschriebenen Schritten sind nachfolgend grau hinterlegt.

```
for (i=1;i<noOfItems;i++) {
        // Arithmetisches Mittel aus den aktuellen Ausprägungspaaren berechnen
        avgItem = ((itemList[i].item)+(itemList[i-1].item))/2;
        // Reinheit/Informationsgewinn der Verteilungen berechnen
        pureness = mIGA->getAttributeIG(avgItem);
```

```
            // Prüfen, ob Reinheit/Informationsgewinn größer als die des Vorgänger-Split-Wertes
            if (pureness > maxPureness) {
                maxPureness = pureness;
                val = avgItem;
            }
    }
```

Abbildung 34: Quellcode der Splitstrategie ohne Verteilungsinformation

Die oben geschilderte Vorgehensweise zur Findung des besten Splits wird in Tabelle 20 anhand eines Beispiels erläutert. Die Abbildung zeigt alle potentiellen Splitpunkte (arithmetisches Mittel) der numerischen Dimension „Alter“ und den dazugehörigen Informationsgewinn bei einem Split am jeweiligen Punkt. Nach der Berechnung des Informationsgewinnes für einen Splitpunkt, wird dieser mit dem Informationsgewinn des aktuell besten Splits verglichen und gegebenenfalls neu gesetzt. Die Gesamtentropie in diesem Beispiel beträgt 0,97095. Der Beste Splitpunkt liegt bei 27,5 mit einem maximalen Informationsgewinn von 0,4199.

ID	Alter	Familienstand	Kinder	Arithmetisches Mittel	Informationsgewinn
2	17	ledig	nein	-	-
1	23	geschieden	nein	(17+23)/2 = 20	0,17095
5	32	verheiratet	ja	(23+32)/2 = 27,5	**0,4199**
3	43	ledig	nein	(32+43)/2 = 37,5	0,0199
4	68	geschieden	ja	(43 + 68)/2 = 55,5	0,3219

Tabelle 20: Splitpunkte bei der Splitstrategie ohne Verteilungsinformation

Die Splitstrategie ohne Verteilungsinformation enthält zwei entscheidende Nachteile:

- Die Attributausprägungen jeder Dimension müssen vor der Evaluierung der Splits sortiert werden.
- Die Splitstrategie untersucht für n Attributausprägungen n-1 Splitpunkte, was bei einer großen Anzahl von Trainingsdaten sehr aufwändig ist. Pro Knoten müssen für k Attribute der Trainingsmenge $k \cdot (n-1)$ Splitpunkte evaluiert werden.

Diese Nachteile wirken sich rein auf die Effizienz des DT-Algorithmus aus. Sie sind aber auch Grund dafür, warum es zweckmäßig ist, Hilfsinformationen einzusetzen,

die das Auffinden von Splitpunkten vereinfachen. Hauptaugenmerk wird weiterhin auf eine Qualitätssteigerung des Klassifikators gelegt.

Min/Max-Strategie

Bei der Min/Max-Strategie (vgl. Kapitel 6.2) werden die Minima und Maxima der Cluster einer Dimension als Splitpunkte evaluiert. Die potentiellen Splits werden nicht wie bei der Splitstrategie ohne Verteilungsinformation aus den Attributausprägungen der Partition des aktuellen Knotens berechnet. Vielmehr wird jedes Minimum bzw. Maximum der Cluster einer Dimension als Splitpunkt angenommen und der Informationsgewinn für einen Split am jeweiligen Punkt berechnet. Die Hilfsinformationen (Minimum, Maximum) beziehen sich nicht auf die Partition des aktuellen Knotens, sondern auf die gesamte Trainingsmenge. Sie werden nicht für jede durch einen Split entstandene Partition neu berechnet, d.h. dass die Splitpunkt-Kandidaten einer Dimension für jeden Knoten gleich sind. Der „optimale“ Split einer Dimension ist jenes Extremum, dessen Informationsgewinn am höchsten ist. Abbildung 35 zeigt einen Auszug aus dem dazugehörigen Quellcode. Das Feld „mmt“ enthält für jeden Cluster einer Dimension das Minimum und das Maximum. Die Kommentare der einzelnen Schritte des Verfahrens sind nachfolgend grau hinterlegt.

```
// Reinheit der Verteilung für jeden Splitpunkt-Kandidaten berechnen
for (i=0;i<cMaxClusters;i++) {
        // Reinheit/Informationsgewinn bei Split im Minimum
        pureness = mIGA->getAttributeIG(mmt[i].m_min);
        // Prüfen, ob Reinheit/Informationsgewinn größer als die des Vorgänger-Split-Wertes
        if (pureness > maxPureness) {
                maxPureness = pureness;
                val = mmt[i].m_min;
        }
        // Reinheit/Informationsgewinn bei Split im Maximum
        pureness = mIGA->getAttributeIG(mmt[i].m_max);
        // Prüfen, ob Reinheit/Informationsgewinn größer als die des Vorgänger-Split-Wertes
        if (pureness > maxPureness) {
                maxPureness = pureness;
                val = mmt[i].m_max;
        }
}
```

Abbildung 35: Quellcode der Min/Max-Splitstrategie

Das folgende Beispiel (vgl. Tabelle 21) zeigt die potentiellen Splits der Dimension „Alter“ und die dazugehörigen Informationsgewinne bei der Min/Max-Splitstrategie. Die Gesamtentropie beträgt 0,97095. Der beste Splitpunkt liegt bei 43 mit einem maximalen Informationsgewinn von 0,3219.

ID	Alter	Familienstand	Kinder	Min/Max	Informationsgewinn
2	17	ledig	nein	MIN(Hoch) = 17	0,17095
1	23	geschieden	nein	-	-
5	32	verheiratet	ja	MIN(Niedrig) = 32	0,0199
3	43	ledig	nein	MAX(Hoch) = 43	**0,3219**
4	68	geschieden	ja	MAX(Niedrig) = 68	0

Tabelle 21: Splitpunkte bei der Min/Max-Splitstrategie

Die Min/Max-Strategie überwindet alle Nachteile der Strategie ohne Verteilungsinformationen. Die Attributausprägungen müssen im Vorhinein nicht sortiert werden, da die Splitpunkte aus der Hilfsinformationstabelle stammen. Die Anzahl der Splitpunkt-Kandidaten ist nicht abhängig von der Zahl der Attributausprägungen einer Dimension, sondern von der Anzahl der Cluster, die im Normalfall sehr viel kleiner ist. Die Anzahl der Splitpunkt-Kandidaten beträgt für eine Dimension mit c Clustern $2 \cdot c$.

Ein Nachteil dieser Strategie könnte die Tatsache sein, dass Minimum und Maximum nicht für jede Partition neu berechnet werden. Für diese Vorgehensweise spricht aber, dass die Hilfsinformationen Metainformation über die Grundgesamtheit sind, aus der die Trainingsdaten stammen. Der Klassifikator könnte daher auf eine genügend große Testmenge aus der Grundgesamtheit eine hohe Klassifikationsgenauigkeit erreichen.

Inwieweit die Hilfsinformationen „Minimum und Maximum der Cluster“ im Detail Einfluss auf die Qualität des Klassifikators nehmen, zeigen die Testergebisse in Kapitel 8.

Dichtefunktion-Strategie

Die Dichtefunktion-Strategie (vgl. Kapitel 6.2) verwendet als Splitpunkt-Kandidaten die Schnittpunkte von Dichtefunktionen der Cluster einer Dimension. Wie bei der Min/Max-Strategie werden nicht die Attributausprägungen, sondern Hilfsinformationen aus einer separaten Tabelle dazu verwendet, Splitpunkte zu

finden bzw. zu evaluieren. Aus den Hilfsinformationen „Mittelwert“ (Erwartungswert) und „Standardabweichung“ eines jeden Cluster einer Dimension wird für jeden Cluster eine Dichtefunktion erzeugt. Für jede Dichtefunktion werden jene Schnittpunkte mit allen anderen Dichtefunktionen berechnet, die zwischen den Mittelwerten von jeweils zwei Clustern liegen („relevante Schnittpunkte“, vgl. Kapitel 6.2). Für k Cluster c1, …, ck existieren, unter Ausschluss der Schnittpunkte mit ck selbst und unter der Bedingung, dass der Schnittpunkt von ck mit ck+1 gleich dem Schnittpunkt von ck+1 mit ck ist, genau

$$S_k = \frac{k \cdot (k-1)}{2}$$

relevante Schnittpunkte (vgl. „Brook´sches Gesetz“ [BH03]).
Die Hilfsinformationen beziehen sich auf die Grundgesamtheit der Daten. Die berechneten Schnittpunkte in einer Dimension gelten für jeden Knoten unabhängig von der Größe seiner aktuellen Partition. Für jeden Schnittpunkt einer Dimension wird der Informationsgewinn berechnet und der Schnittpunkt mit dem höchsten Wert wird als optimaler Splitpunkt dieser Dimension angenommen. Abbildung 36 illustriert die Vorgehensweise bei der Dichtefunktion-Strategie anhand eines Auszugs aus dem Quellcode. Die Datenstruktur „mst“ enthält für jeden Cluster einer Dimension Mittelwert und Standardabweichung. Die Schritte der Berechnung der Splitpunkte bei der Dichtefunktion-Strategie sind im Quellcode grau hinterlegt.

```
for (i=0;i<e.getNoOfClusters()-1;i++) {
        for (j=1;j<e.getNoOfClusters();j++) {
                // Keine Schnittpunkte mit demselben Cluster und mit bereits durchlaufenen
                Clustern berechnen
                if (i < j) {
                        // Schnittpunkt der Cluster berechnen
                        currIntersection = fabs((mst[j].m_dev * mst[i].m_mean - mst[i].m_dev *
                        mst[j].m_mean) / (mst[j].m_dev - mst[i].m_dev));
                        // Prüfen, ob Schnittpunkt zwischen den Erwartungswerten der Cluster
                        liegt - wenn nicht, dann Neuberechnung des Splitwertes
                        if (((currIntersection >= mst[i].m_mean) && (currIntersection >=
                        mst[j].m_mean)) || ((currIntersection < mst[i].m_mean) &&
                        (currIntersection < mst[j].m_mean))) {
```

```
                                currIntersection = fabs((mst[j].m_dev * mst[i].m_mean +
                                mst[i].m_dev * mst[j].m_mean) / (mst[j].m_dev +
                                mst[i].m_dev));
                        }
                        // Schnittpunkt speichern – es existieren maximal k = cMaxClusters *
                        (cMaxClusters-1) / 2 „relevante“ Schnittpunkte
                        intersectionX[k++] = currIntersection;
                }
        }
}

// Reinheit der Verteilung für jeden Splitpunkt-Kandidaten berechnen
for (i=0;i<k;i++) {
        // Reinheit/Informationsgewinn der Verteilungen berechnen
        pureness = mIGA->getAttributeIG(intersectionX[i]);
        // Prüfen, ob Reinheit/Informationsgewinn größer als die des Vorgänger-Split-Wertes
        if (pureness > maxPureness) {
                maxPureness = pureness;
                val = intersectionX[i];
        }
}
```

Abbildung 36: Quellcode der Dichtefunktion-Splitstrategie

Im folgenden Beispiel (vgl. Tabelle 22) werden die potentiellen Splitpunkte der Dimension „Alter“ und der korrespondierende Informationsgewinn mittels der Dichtefunktion-Strategie ermittelt. Die Gesamtentropie der Tabellenpartition beträgt 0,97095. Der beste Splitpunkt liegt bei 36,195 mit einem Informationsgewinn von In diesem Fall sind die Daten nur zwei Klassen zugeordnet, weshalb lediglich ein relevanter Schnittpunkt existiert.

ID	Alter	Familien-stand	Kinder	Mittelwert	Standardab-weichung	Schnittpunkt	Informations-gewinn
1	23	geschieden	nein	27,67	11,12	Irrelevant: -8,43023	0
2	17	ledig	nein				
3	43	ledig	nein			**Relevant: 36,1950055**	**0,0199**
4	68	geschieden	ja	50	18		
5	32	verheiratet	ja				

Tabelle 22: Splitpunkte bei der Dichtefunktion-Strategie

Die Dichtefunktion-Strategie arbeitet effizienter als die Splitstrategie ohne Verteilungsinformation. Sie verzichtet auf ein Sortieren der Partitionsdaten und

evaluiert ein Maximum von k·(k-1)/2 Splitpunkt-Kandidaten. Ein weiterer Vorteil könnte darin liegen, dass die Schnittpunkte die Cluster möglichst mittig trennen. Der Fall, dass wie bei der Min/Max-Strategie wahllos durch einen Cluster hindurch geschnitten wird ist unwahrscheinlicher. Die Dichtefunktion-Strategie garantiert, dass zumindest zwei Cluster möglichst rein von einander getrennt werden.
Welchen Einfluss die Hilfsinformationen im Detail auf die Qualität des Klassifikators nehmen, wird in Kapitel 8 anhand der Testergebnisse erläutert.

7.2.3 Konstruktion des Entscheidungsbaums

Die Konstruktion des Entscheidungsbaumes läuft rekursiv ab. Numerische Attribute werden binär gesplittet, kategorische n-är entsprechend der Anzahl der verschiedenen Attributausprägungen einer Dimension. Die Konstruktion beginnt mit dem Wurzelknoten (root), für den vor dem rekursiven Prozess seine AVC-Gruppe aus der gesamten Trainingstabelle erstellt wird. Um den rekursiven Aufbau zu starten, wird die Methode „buildDT()“ für den Wurzelknoten aufgerufen. Innerhalb dieser Methode wird zuerst aus der AVC-Gruppe des Knotens jenes Attribut und dessen Splitpunkt bestimmt, das den größten Informationsgewinn beim Split liefert. Die numerischen Attribute besitzen einen berechneten Splitpunkt, während kategorische Attribute anhand ihrer Attributausprägung einem Knoten zugeordnet werden.
Nach der Bestimmung des Split-Attributs und des Splitwertes, werden in Abhängigkeit des Typs des Attributs (numerisch, kategorisch) rekursiv die Sohnknoten erzeugt. Für jeden Sohnkoten wird im Vorfeld eine Datenbankpartition im Hinblick auf das Splitkriterium (Splitpunkt oder Ausprägung) erstellt. Aus der Partition kann dann für den Sohnknoten seine AVC-Gruppe erzeugt und der rekursive Aufruf durchgeführt werden. Abbildung 37 zeigt einen Auszug aus dem Quellcode der Methode „buildDT“. Die einzelnen Schritte der Entscheidungsbaum-Konstruktion sind im Quellcode grau hinterlegt.

```
void Node::buildDT() {
        allParents = true;
        // Informationsgewinn der Splitwerte jedes Attributs berechnen und optimalen Split
        evaluieren
```

```
(1)     for (int i=0;i<e.getNoOfAttributes()-1;i++) {
            // Setzen des aktuellen AVC-Sets
            avc = avcGroup.groupOfAVCSets[i];
            // Sicherstellen, dass aktuelles Split-Attribut != dem Split-Attribut eines
            Vaterknotens ist
            if (!isParent(avc->getAttribute())) {
                // Das Attribut ist nicht Split-Attribut eines Vaterknotens
                allParents = false;
                // Erzeugen der Split-Kriterien (values) des Knotens, 0… numerisch
                if (avc->getAttribute()->getAType() == 0) {
                    s = &ns;
                }
                else { // 1...kategorisch
                    s = &cs;
                }
                // Informationsgewinn berechnen
                nextInfoGain = s->getInfoGain(avc,partitionName);
                // Prüfen des neuen Informationsgewinns und Setzen des
                Splitkriteriums
                if (nextInfoGain > maxInfoGain) {
                    // Maximaler Informationsgewinn
                    maxInfoGain = nextInfoGain;
                    // AVC-Set mit dem größten Informationsgewinn
                    maxIGAttr = avc;
                    // Setzen des Splitkriteriums
                    splitCriteria = s;
                }
            }
(2)     }

        // Baum erzeugen
(3)     // Abbruchkriterium
        if ((maxIGAttr->getNoOfClassifiers() == 1) || (maxIGAttr->getSupport() <=
        cMinSupport) || (maxIGAttr->getConfidence() >= cMinConfidence) || (nodeNumber >=
        cMaxPartitions) || (allParents)) {
            // Anzahl der Ausprägungen am Knoten <= cMinSupport oder alle
            Ausprägungen fallen in eine Klasse
            if (maxIGAttr->getNoOfClassifiers() == 1) {
                // Blattknoten-Klasse ist einzige Klasse aus dem maxIGAttr-AVC-Set
                setLeaveValue(maxIGAttr->avcSet->avcSubSet.classifier);
            }
```

```
                else {
                        // mehr als eine Klasse am Knoten (minSupport unterschritten,
                        minConfidence übertroffen, nodeNumber >= cMaxPartitions)
                        // Blattknoten-Klasse ist jene Klasse, der die meisten Ausprägungen
                        am Knoten angehören
                        setLeaveValue(maxIGAttr->getMaxValueId());
                }
(4)     }
(5)     else { // Erstellen der Sohnknoten und der NEUEN Datenbank-Partitionen
                if (splitAttribute->getAType() == 0) { // numerischer Split (binär)
                        // Sohnknoten mit Werten ≤ Splitkriterium
                        listOfSons[0] = new Node();
                        // Setzen aller Attribute des Sohnknotens und der AVC-Gruppe
                        …
                        // Alle Sub-Trees durchlaufen (Rekursion)
                        listOfSons[0]->buildDT();
                        // Sohnknoten mit Werten > Splitkriterium
                        listOfSons[1] = new Node();
                        // Setzen aller Attribute des Sohnknotens und der AVC-Gruppe
                        …
                        // Alle Sub-Trees durchlaufen (Rekursion)
                        listOfSons[1]->buildDT();
(6)             }
(7)             else { // kategorischer Split
                        // Erstellen einer Partition für jede Attributausprägung
                        for (i=0;i<maxIGAttr->getNoOfValOccurrence();i++) {
                                listOfSons[i] = new Node();
                                // Setzen aller Attribute des Sohnknotens und der AVC-
                                Gruppe
                                …
                                // Alle Sub-Trees durchlaufen (Rekursion)
                                listOfSons[i]->buildDT();
                        }
(8)             }
        }
}
```

Abbildung 37: Quellcode der Entscheidungsbaum-Konstruktion

In Abbildung 34 ist in den Zeilen (1) bis (2) die Identifikation des Splitattributs und des dazugehörigen besten Splits ersichtlich. Die Abbruchkriterien und die Erstellung

der Blattknoten befinden sich in den Zeilen (3) bis (4). Die Zeilen (5) bis (6) implementieren die Erzeugung eines numerischen Splits, während die Zeilen (7) bis (8) einen kategorischen Split darstellen.
Das Abbruchkriterium (siehe Zeile (3)) setzt sich aus mehreren Bedingungen zusammen. Ein Blattknoten wird immer dann aus dem aktuellen Knoten erstellt, wenn eine der folgenden Bedingungen zutrifft:

- Die Partition des Knotens (die AVC-Gruppe) enthält nur noch Tupel einer einzigen Klasse.

- Die Partition des Knotens überschreitet die Maximalanzahl von Partitionen. Diese ist als Konstante „cMaxPartitions = 100“ festgelegt.

- Das Split-Attribut des aktuellen Knotens ist Split-Attribut eines Vaterknotens oder der aktuelle Knoten hat bereits alle Attribute als Split-Attribute der Vaterknoten.

- Der minimale Support „cMinSupport = 50“ wird durch die Partition des aktuellen Knotens unterschritten, d.h. es existieren weniger als cMinSupport Tupel in der Partition.

- Die minimale Konfidenz „cMinConfidence = 0.6“ (60%) wird durch die Partition des aktuellen Knotens überschritten, d.h. dass in der Partition Tupel enthalten sind, deren Klasse mehr als 60% aller Klassen der Partition ausmacht.

Nachdem der Entscheidungsbaum im Hauptspeicher erstellt wurde, wird er als Baumtabelle zurück in die Datenbank geschrieben. Anhand der Baumtabelle werden anschließend die Testdaten in ihre entsprechende Klasse eingeordnet. Die Tabelle des Baumes besitzt folgende Form (vgl. [SD01]):

BAUMTABELLE							
KnotenNr	**VaterknotenNr**	**KnotenTyp**	**SplitAttribut**	**Vaterknoten-SplitAttribut**	**Minimaler Splitwert**	**Maximaler Splitwert**	**Klasse**
0	-	root	Autotyp	-	-	-	Hoch
1	0	internal	Alter	Autotyp	0	1..Familie	Niedrig
2	0	internal	Alter	Autotyp	1	2..Sport	Hoch
…	…	…	…	…	…	…	…
5	2	leaf	Versicherung	Alter	MIN(Alter)	37,5	Hoch
…	…	…	…	…	…	…	…
k	…	…	…	…	…	…	…

Tabelle 23: Repräsentation des Entscheidungsbaumes in der Datenbank

Für jedes Testtupel werden im Vorfeld nur jene Einträge der Baumtabelle zum Vergleich selektiert, die bei der Bestimmung der Klasse nötig sein könnten. Es werden all jene Knoten selektiert, deren Splitkriterien von den Attributausprägungen des jeweiligen Testtupels erfüllt werden. Die benötigten Knoten für ein Testtupel T mit den Attributen (A_1, …, A_n) und den Attributausprägungen (a_1, …, a_n) unterliegen folgender Datenbankabfrage (vgl. [SD01]):

SELECT * **FROM** BAUMTABELLE
WHERE (Vaterknoten-SplitAttribut = A_1 **AND** Minimaler_Splitwert < a_1 **AND** Maximaler_Splitwert ≥ a_1) **OR**
(Vaterknoten-SplitAttribut = A_2 **AND** Minimaler_Splitwert < a_2 **AND** Maximaler_Splitwert ≥ a_2) **OR**
…
(Vaterknoten-SplitAttribut = A_n **AND** Minimaler_Splitwert < a_n **AND** Maximaler_Splitwert ≥ a_n)

Die aus der obigen Query entstehenden Kandidaten-Knoten werden dann zur Bestimmung der Klasse eines Testtupels herangezogen. Wie die Kandidatenknoten bei der Klassenzuordnung eingesetzt werden zeigt ein Auszug aus der Implementierung des durch [SD01] vorgestellten „Prediction Join“ in Abbildung 38. Die Kandidatenknoten werden in einer temporären Knotentabelle abgelegt. Beim Wurzelknoten beginnend wird solange ein Sohnknoten mit dem aktuellen Knoten als Vaterknoten gesucht, bis kein Knoten mehr Sohn des aktuellen Knotens ist. Die

Klasse des aktuellen Knotens wird dem Testtupel als Klasse zugewiesen. Die Vorgehensweise beim „Prediction Join“ ist grau hinterlegt kommentiert.

```
// Lesen von KnotenNummer, VaterNumer und Klasse des 1. Knotens aus der Datenbank
EXEC SQL FETCH ntable_cursor INTO :node_number,:parent_number,:class_number;
currNode = (int)node_number;
currParent = (int)parent_number;
currClass = (int)class_number;
finished = false;
// Bestimme Weg des Testtupels durch den Baum
do {
        do {
                EXEC SQL FETCH ntable_cursor INTO :node_number, :parent_number,
                :class_number;
                if (sqlca.sqlcode==1403) { // Kein Knoten mehr gefunden
                        // Aktuelles Tupel inkl. berechnete Klasse in resultTable einfügen
                        INSERT INTO resultTable…
                        // Vaterknoten unbestimmt und Suche der Klasse abgeschlossen
                        currParent = -1;
                        finished = true;
                }
                else { // Knoten gefunden
                        currParent = (int)parent_number;
                }
        } while ((currParent != currNode) && (currParent != -1)); // wiederhole, solange
        aktueller Vaterknoten != aktueller Knoten ODER Vaterknoten unbestimmt ist
        // Knotennummer und Klasse auf nächsten Knoten setzen
        currNode = (int)node_number;
        currClass = (int)class_number;
} while (!finished); // wiederhole, solange ein Knoten gefunden wird
```

Abbildung 38: Quellcode des „Prediction Join“

Die aus der Durchführung des „Prediction Join“ entstehende Ergebnistabelle (resultTable) enthält die Testtupel mit all ihren Attributen und eine zusätzliche Spalte für die vom Klassifikator bestimmte Klasse (PREDICTED_CLASS). Neben den Attributen befinden sich die Spalten REAL_CLUSTER_ID und ESTIMATED_CLUSTER_ID von Beginn an in der resultTable. Sie enthalten ebenfalls Klassenzugehörigkeiten der Testtupel. Die REAL_CLASS wurde von einem Testdatengenerator zugewiesen. Die ESTIMATED_CLUSTER_ID wurde vom

Vorgängerverfahren „K-Means“ bestimmt. Alle drei Spalten, die Klassenzugehörigkeiten beinhalten, werden zur Berechnung der Qualitätskriterien des Entscheidungsbaum-Klassifikators verwendet.

7.2.4 Berechnen der Qualitätskriterien

Nachdem jedem Testtupel vom Klassifikator eine Klasse zugewiesen wurde, werden mithilfe der Ergebnistabelle die Qualitätskriterien berechnet. Dazu werden die Spalten REAL_CLUSTER_ID, ESTIMATED_CLUSTER_ID und PREDICTED_CLASS aus der Ergebnistabelle selektiert und „verglichen“. Folgende Qualitätskriterien werden berechnet (siehe Kapitel 3.2):

- **Qualitätskriterien in Bezug auf die Relevanz der Testdaten**
    - Präzision (precision)
    - Vollständigkeit (recall)
    - Genauigkeit (accuracy)
- **Qualitätskriterien unabhängig von der Relevanz der Testdaten**
    - Klassifikationsgenauigkeit
    - Tatsächlicher Klassifikationsfehler
    - Beobachteter Klassifikationsfehler (bei Test auf Trainingsdaten)

Die Qualitätskriterien bzgl. der Relevanz der Testdaten treffen darüber hinaus eine Aussage hinsichtlich der Qualität des Ergebnisses des Clusteringverfahrens „K-Means“. Die Precision z.B. stellt die relevanten und richtig klassifizierten Testtupel allen richtig klassifizierten Tupeln gegenüber. Ist dieser Wert sehr klein, so ist die Wahrscheinlichkeit sehr groß, dass das Clustering viele von der REAL_CLUSTER_ID abweichende Einträge produziert hat. Eine detaillierte Analyse der Qualitätskriterien wird in Kapitel 8 im Zuge des Vergleichs der Testergebnisse durchgeführt.

Für die Berechnung der Qualitätskriterien bzgl. der Relevanz müssen die klassifizierten Testtupel in vier Mengen unterteilt werden. Die Häufigkeiten der Testtupel, die diesen Mengen angehören, wurden in Kapitel 3.2 mit a, b, c und d bezeichnet. Sie besitzen folgenden Hintergrund:

- Die Häufigkeit a ist die Anzahl aller relevanten, richtig klassifizierten Testtupel.
- Die Häufigkeit c ist die Anzahl aller relevanten, falsch klassifizierten Testtupel.
- Die Häufigkeit b ist die Anzahl aller irrelevanten, richtig klassifizierten Testtupel.
- Die Häufigkeit d ist die Anzahl aller irrelevanten, falsch klassifizierten Testtupel.

Zur Berechnung der von der Relevanz unabhängigen Qualitätskriterien reicht es aus, alle Testtupel in richtig oder falsch klassifizierte einzuteilen. Aus diesen Häufigkeiten wird bestimmt, inwieweit das Ergebnis des Entscheidungsbaum-Klassifikators mit dem des Clusterings übereinstimmt.
Der in Abbildung 39 ersichtliche Auszug aus dem Quellcode zeigt die Berechnung der Qualitätskriterien unter Verwendung der erläuterten Häufigkeiten. Die einzelnen Schritte bei der Berechnung und die Kommentare sind grau hinterlegt.

```
// Lesen der Klassenzugehörigkeiten aus der Ergebnistabelle
EXEC SQL FETCH cerror_cursor INTO :class0,:class1,:class2;
// Bestimmen der Qualitätskriterien für alle Testtupel
while (sqlca.sqlcode!=1403) {
        r_class = (int)class0;
        e_class = (int)class1;
        p_class = (int)class2;
        // Bestimmen der Häufigkeiten bzgl. der Relevanz
        if (r_class == e_class) { // relevant
                if (e_class == p_class) { // relevant, richtig klassifiziert
                        a++;
                }
                else { // relevant, falsch klassifiziert
                        c++;
                }
        }
        else { // irrelevant
```

```
                    if (e_class == p_class) { // irrelevant, richtig klassifiziert
                        b++;
                    }
                    else { // irrelevant, falsch klassifiziert
                        d++;
                    }
                }
                // Lesen der Klassenzugehörigkeiten des nächsten Testtupels
                EXEC SQL FETCH cerror_cursor INTO class0,:class1,:class2;
            }
            // Bestimmen der Häufigkeiten unabhängig von der Relevanz
            valOccAll = a + b + c + d;                          // alle Testtupel
            measures = a + b;                                   // richtig Klassifizierte Testtupel
            errors = valOccAll - measures;                      // falsch Klassifizierte Testtupel
            // Berechnen der Qualitätskriterien
            precision = (float)a / (float)(a + b);              // Präzision
            recall = (float)a / (float)(a + c);                 // Vollständigkeit
            accuracy = (float)(a + d)/(float)valOccAll;         // Genauigkeit
            cl_accuracy = (float)measures / (float)valOccAll;   // Klassifikationsgenauigkeit
            true_error = (float)errors / (float)valOccAll;      // Tatsächlicher Klassifikationsfehler
(1)         apparent_error = -1;                                // Beobachteter Klassifikationsfehler
(2)         apparent_error = (float)errors / (float)valOccAll;  // Beobachteter Klassifikationsfehler
```

Abbildung 39: Quellcode der Bestimmung der Qualitätskriterien

Der „beobachtete Klassifikationsfehler“ (apparent_error) liefert nur bei einem Test über die Trainingsdaten des Klassifikators ein aussagekräftiges Ergebnis. Bei einem Test über eine zur Trainingsmenge disjunkte Testmenge liefert der apparent_error den Wert -1 zurück (siehe Quellcode Zeile (1)). Wird der Test über die Trainingsmenge durchgeführt, so entspricht der apparent_error den „tatsächlichen Klassifikationsfehler“ (siehe Quellcode Zeile (2)).

Das nun folgende Kapitel erläutert ausgewählte Testreihen, die Ergebnisse der Tests und den Einfluss der verwendeten Hilfsinformationen auf die Qualität des Klassifikators.

8 Testreihen und Ergebnisse

Dieses Kapitel erläutert zunächst die Ausgangsdaten, aus denen die Trainings- und Testdaten für den Entscheidungsbaum-Klassifikator stammen. Anschließend werden die durchgeführten Testreihen mit allen Kombinationen von verwendeten Hilfsinformationen aus dem Vorgängerverfahren beschrieben. Abschließend werden die Testergebnisse in Tabellenform bzw. grafisch dargestellt und die verwendeten Hilfsinformationen hinsichtlich ihrer Brauchbarkeit analysiert. Die Testläufe sollen die Qualitätsunterschiede zwischen den erstellten Klassifikatoren bei der Verwendung von Hilfsinformationen zeigen. Zur Messung der Qualität werden die Klassifikationsgenauigkeiten von Klassifikatoren, die ohne bzw. mit Hilfsinformationen erzeugt wurden, herangezogen. Eine Untersuchung hinsichtlich einer Effizienzsteigerung bzw. eines Effizienzverlustes ist nicht Ziel der Tests. Die Verringerung der Laufzeit durch die Verwendung von Hilfsinformationen ist ein positiver Nebeneffekt.

8.1 Trainings- und Testdaten

Als Ausgangsdaten dienen Clusteringergebnisse (Tabelle „clusteringtestdaten“), die in einer parallelen Diplomarbeit [SK04] aus dem Vorgängerverfahren „K-Means“ erzeugt wurden. Diese Ausgangsdaten bestehen aus zehn numerischen, fünf kategorischen Dimensionen und einer Spalte, welche die Klasse (REAL_CLUSTER_ID) eines Datensatzes enthält. Die REAL_CLUSTER_ID wurde von einem Datengenerator [GM04] zugewiesen, der die initiale Verteilung der Ausgangsdaten bewirkte. Die folgende Tabelle 24 stellt die Ausgangsdaten in Form ihrer Dimensionen dar. Numerische Attribute sind durchwegs vom Datentyp FLOAT, während kategorische Attribute den Datentyp NUMBER besitzen.

Name	Typ	Beschreibung
X1	FLOAT(126)	Numerische Dimension 1
X2	FLOAT(126)	Numerische Dimension 2
X3	FLOAT(126)	Numerische Dimension 3
X4	FLOAT(126)	Numerische Dimension 4
X5	FLOAT(126)	Numerische Dimension 5
X6	FLOAT(126)	Numerische Dimension 6
X7	FLOAT(126)	Numerische Dimension 7
X8	FLOAT(126)	Numerische Dimension 8
X9	FLOAT(126)	Numerische Dimension 9
X10	FLOAT(126)	Numerische Dimension 10
K1	NUMBER(2)	Kategorische Dimension 1
K2	NUMBER(2)	Kategorische Dimension 2
K3	NUMBER(2)	Kategorische Dimension 3
K4	NUMBER(2)	Kategorische Dimension 4
K5	NUMBER(2)	Kategorische Dimension 5
REAL_CLUSTER_ID	NUMBER(2)	Ermittelte Cluster-ID des Datengenerators
ESTIMATE_CLUSTER_ID	NUMBER(2)	Ermittelte Cluster-ID durch K-Means bei fünf numerischen und fünf kategorischen Dimensionen
ESTIMATE_CLUSTER_ID_10	NUMBER(2)	Ermittelte Cluster-ID durch K-Means bei zehn numerischen Dimensionen

Tabelle 24: Aufbau der Ausgangsdaten-Tabelle „clusteringtestdaten“

Für das Training der in dieser Arbeit benötigten Entscheidungsbäume werden jene Klassen bzw. Clusternummern verwendet, die durch K-Means erzeugt wurden. Die Spalten, die jene für die Entscheidungsbaumkonstruktion notwendigen Klassen beinhalten, sind mit ESTIMATE_CLUSTER_ID und ESTIMATE_CLUSTER_ID_10 bezeichnet. Sie werden auch Zielattribute genannt und sind in Tabelle 24 ersichtlich. Um die Ausgangsdaten zu veranschaulichen, werden in Abbildung 40 die Dimensionen „X1“ und „X2“ aufgetragen. Diese beiden Dimensionen repräsentieren die Verteilung der Cluster in den Ausgangsdaten besonders gut [GM04].

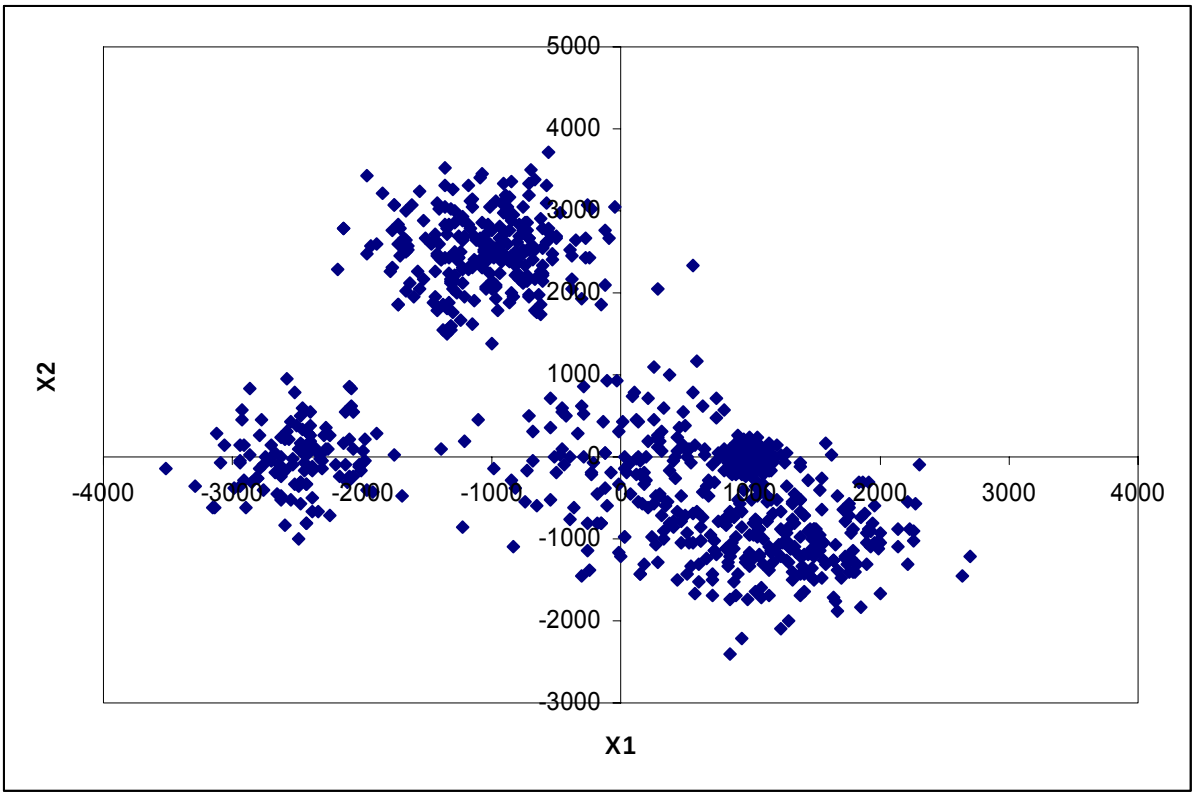


Abbildung 40: Ausgangsdaten – Dimensionen X1 und X2 aus „clusteringtestdaten“

Aus den Clusteringergebnissen wurden zwei Teilmengen selektiert, die einerseits aus fünf numerischen und fünf kategorischen Dimensionen bestehen (testdata_5_5) und andererseits aus zehn numerischen Dimensionen zusammengesetzt sind (testdata_10_0). Die Tabelle testdata_5_5 enthält die Clusteringergebnisse der Testreihe 4 aus [SK04] mit 100.000 Tupeln und dem Zielattribut ESTIMATE_CLUSTER_ID_10. Die Tabelle testdata_10_0 enthält die Clusteringergebnisse der Testreihe 5 aus [SK04] mit 100.000 Tupeln und dem Zielattribut ESTIMATE_CLUSTER_ID. Darüber hinaus dienen vier weitere Tabellen als Ausgangsdaten, die für beide Teilmengen die den Clustermittelpunkten am nächsten gelegenen und am weitesten entfernten Punkte beinhalten. Sie enthalten jeweils 6.000 Tupel mit repräsentativen Daten (nahe und entfernte Punkte) und wurden im Zuge des Vorgängerverfahrens „K-Means“ erzeugt.

Im Folgenden sind die Tabellennamen der Ausgangsdaten und die Datenbankabfragen zur Erstellung der Ausgangstabellen ersichtlich:

```
-- Ausgangstabelle mit fünf numerischen und fünf kategorischen Attributen
CREATE TABLE testdata_5_5 AS
        (SELECT x1,x2,x3,x6,x9,k1,k2,k3,k4,k5,real_cluster_id,estimate_cluster_id_10
```

```
        FROM stoett.clusteringtestdaten WHERE estimate_cluster_id_10 > 0);
-- Ausgangstabelle mit den Centroiden nahen Punkten aus testdata_5_5
stoett.kmeans_nearest_5_5_1
--Ausgangstabelle mit den Centroiden entfernten Punkten aus testdata_5_5
stoett.kmeans_farest_5_5_1

-- Ausgangstabelle mit zehn numerischen Attributen
CREATE TABLE testdata_10_0 AS
        (SELECT x1,x2,x3,x4,x5,x6,x6,x8,x9,x10,real_cluster_id,estimate_cluster_id
         FROM stoett.clusteringtestdaten estimate_cluster_id > 0);
-- Ausgangstabelle mit den Centroiden nahen Punkten aus testdata_10_0
stoett.kmeans_nearest_10_0_1
--Ausgangstabelle mit den Centroiden entfernten Punkten aus testdata_10_0
stoett.kmeans_farest_10_0_1
```

Trainingsdaten

Die Trainingsdaten stellen eine repräsentative Stichprobe (Sample) der Ausgangsdaten dar. Zum Trainieren der Entscheidungsbäume wurden sechs Trainings-Sets generiert, die aus jeweils 1.000 Tupel bestehen. Die Stichproben wurden mit der ORACLE-Funktion SAMPLE() erstellt. Jede Trainingstabelle besitzt eine eindeutige Bezeichnung folgender Form.

TRAIN_XXXX_Y1_Y2_ZZZZ_I

TRAIN... Kennzeichen einer Trainingstabelle
XXXX... Ursprung der Testdaten - Zufallspunkte (DB), nahe Punkte (NEAR), entfernte Punkte (FAR)
Y1… Anzahl der numerischen Attribute der Testtabelle
Y2... Anzahl kategorischen Attribute der Testdaten
ZZZZ... Anzahl der Tupel der Testdaten
I... laufende Nummer der Stichprobe

Die sechs Trainingstabellen besitzen folgende eindeutige Bezeichnungen:

```
-- Sample von 1000 Tupeln aus testdata_5_5
train_db_5_5_1000_1
-- Sample von 1000 Tupeln aus stoett.kmeans_nearest_5_5_1
train_near_5_5_1000_1
```

-- Sample von 1000 Tupeln aus stoett.kmeans_farest_5_5_1
train_far_5_5_1000_1

-- Sample von 1000 Tupeln aus testdata_10_0
train_db_10_0_1000_1
-- Sample von 1000 Tupeln aus stoett.kmeans_nearest_10_0_1
train_near_10_0_1000_1
-- Sample von 1000 Tupeln aus stoett.kmeans_farest_10_0_1
train_far_10_0_1000_1

Die Trainingsdaten werden in Form einer Tabellen-Query zur Laufzeit des DT-Algorithmus übergeben.

Testdaten

Für die Qualitätstests der Entscheidungsbaum-Klassifikatoren werden Testdaten per Zufallsstichprobe aus den Ausgangsdaten ermittelt und in Sets zu 100, 200, 300, 500, 1000, 2000, 3000 und 5000 unterteilt. Die Stichproben wurden ebenfalls mit der ORACLE-Funktion SAMPLE() erzeugt. Aus den Tabellen, die nahe und entfernte Punkte zu den Clustern enthalten, wurden keine Testdaten-Samples erzeugt. Sie dienen lediglich als Trainingsdaten, da ihr Einfluss auf die Qualität des Klassifikators bestimmt werden soll. Die Samples der Trainings- bzw. Testdaten besitzen keine Schnittmenge, sie sind disjunkt. Jede Testtabelle hat einen eindeutigen Bezeichner folgender Form:

TEST_XXXX_Y1_Y2_ZZZZ_I

TEST... Kennzeichen einer Testtabelle
XXXX... Ursprung der Testdaten - Zufallspunkte (DB), nahe Punkte (NEAR), entfernte Punkte (FAR)
Y1… Anzahl der numerischen Attribute der Testtabelle
Y2... Anzahl kategorischen Attribute der Testdaten
ZZZZ... Anzahl der Tupel der Testdaten
I... laufende Nummer der Stichprobe

Der DT-Algorithmus bekommt die Testdaten zur Laufzeit durch den Namen der Testtabelle übergeben.

Hilfsinformationen

Bei der Generierung der Entscheidungsbaum-Klassifikatoren werden zwei Kategorien von Hilfsinformationen eingebunden (vgl. Kapitel 6.2).

- Hilfsinformationen – Lageeigenschaften der Trainingsdaten (Randpunkte der Cluster)
- Hilfsinformationen – Verteilungsinformationen zu den Clustern (berechnete Hilfsinformationen)

Als berechnete Hilfsinformationen dienen jene Werte die in der Arbeit von Klaus Stöttinger [SK04] identifizierten und berechnet wurden. Die bei der Erstellung der Entscheidungsbaum-Klassifikatoren verwendeten Hilfsinformationen sind Informationen zu den Clustern. Sie sind in folgenden Tabellen gespeichert:

```
-- Hilfsinformationstabelle zu testdada_5_5
CREATE TABLE help_5_5_1 AS (SELECT * FROM stoett.kmeans_cluster_5_5_1);
-- Hilfsinformationstablle zu testdata_10_0
CREATE TABLE help_10_0_1 AS (SELECT * FROM stoett.kmeans_cluster_10_0_1);
```

Die berechneten Hilfsinformationen werden bei der Verwendung einer Splitstrategie, die Splitpunkte mittels Hilfsinformationen ermittelt, über den Namen der Hilfsinformationstabelle in den DT-Algorithmus eingebunden.
Hilfsinformationen, welche die Lageeigenschaften der Trainingsdaten in den Clustern betreffen, sind speziell erzeugte Tabellen, die Tupel enthalten, die entweder sehr nahe am Clustermittelpunkt oder im äußeren Randbereich des Clusters liegen. Im Unterschied zu den berechneten Hilfsinformationen, welche die Bestimmung der Splitpunkte unterstützen, wirken sich diese lagebezogenen Hilfsinformationen auf die Genauigkeit des Klassifikators und die Toleranz gegenüber Ausreißer-Daten aus. Die lagebezogenen Hilfsinformationen fließen als Metainformation über die Trainingsdaten ein.

8.2 Testreihen und Testergebnisse

Die Tests über die erzeugten Entscheidungsbäume zielen darauf ab, Qualitätsunterschiede zwischen den Klassifikatoren zu bestimmen. Diese Qualitätsunterschiede werden in Abhängigkeit von den für die Konstruktion der Bäume verwendeten Hilfsinformationen einander gegenübergestellt. So lässt sich herausfinden, welche Hilfsinformationen welchen Einfluss auf die Qualität des Klassifikators nehmen. Es soll darüber hinaus festgestellt werden, welcher Qualitätsunterschied sich zwischen einer Ausführung ohne Hilfsinformationen und einer Ausführung mit Hilfsinformationen einstellt.

Die Testphase des DT-Algorithmus besteht aus sechs Testreihen unterteilt nach dem Typ der Trainingsdaten. Diese gliedern sich wie folgt:

- 5 numerische Dimensionen + 5 kategorische Dimensionen
    - Zufällige Punkte
    - Nahe Punkte
    - Entfernte Punkte

- numerische Dimensionen
    - Zufällige Punkte
    - Nahe Punkte
    - Entfernte Punkte

In jeder Testreihe werden drei Entscheidungsbaum-Klassifikatoren (ein Baum pro Splitstrategie) erstellt. Auf jeden Baum werden acht Tests durchgeführt, also ein Testlauf pro Testdaten-Sample (siehe Kapitel 8.1). Zusätzlich wird ein Test über die Trainingsmenge des Entscheidungsbaumes selbst durchgeführt, um den „beobachteten Klassifikationsfehler“ zu ermitteln. Die gesamte Testphase besteht somit aus 6 x 3 x 9 = 162 Testläufen.
Für jede Testreihe fließen unterschiedliche Parameter in den DT-Algorithmus ein. Die Entscheidungsbäume werden mit unterschiedlichen Trainingsdaten erstellt. Die Tests werden jedoch für jeden Baum mit den gleichen Testdaten-Samples durchgeführt.

Die nachfolgende Tabelle 24 zeigt die unterschiedlichen Parameter für jede Testreihe:

Parameter	**Testreihe 1**	**Testreihe 2**	**Testreihe 3**
Username	humer	humer	humer
Passwort	******	******	******
SID	dke3	dke3	dke3
Trainingstabellen	train_db_5_5_1000_1	train_near_5_5_1000_1	train_far_5_5_1000_1
Zielattributname	estimate_cluster_id_10	estimate_cluster_id_10	estimate_cluster_id_10
Minimaler Support	50 (5%)	50 (5%)	50 (5%)
Minimale Konfidenz	60%	60%	60%
Splitparameter	1, 2, 3	1, 2, 3	1, 2, 3
Hilfsinformations-tabelle	help_5_5_1	help_5_5_1	help_5_5_1
Baumtabellen	tree_db_5_5_1000_1_1, tree_db_5_5_1000_2_1, tree_db_5_5_1000_3_1	tree_near_5_5_1000_1_1, tree_near_5_5_1000_2_1, tree_near_5_5_1000_3_1	tree_far_5_5_1000_1_1, tree_far_5_5_1000_2_1, tree_far_5_5_1000_3_1
Testtabellen	test_db_5_5_100, …, test_db_5_5_5000	test_db_5_5_100, …, test_db_5_5_5000	test_db_5_5_100, …, test_db_5_5_5000
Input-Datei	input_db_5_5	input_near_5_5	input_far_5_5
Output-Datei	output_db_5_5	output_near_5_5	output_far_5_5
	Testreihe 4	**Testreihe 5**	**Testreihe 6**
Username	humer	humer	humer
Passwort	******	******	******
SID	dke3	dke3	dke3
Trainingstabellen	train_db_10_0_1000_1	train_near_10_0_1000_1	train_far_10_0_1000_1
Zielattributname	estimate_cluster_id	estimate_cluster_id	estimate_cluster_id
Minimaler Support	50 (5%)	50 (5%)	50 (5%)
Minimale Konfidenz	60%	60%	60%
Splitparameter	1, 2, 3	1, 2, 3	1, 2, 3
Hilfsinformations-tabelle	help_10_0_1	help_10_0_1	help_10_0_1
Baumtabelle	tree_db_10_0_1000_1_1, tree_db_10_0_1000_2_1, tree_db_10_0_1000_3_1	tree_near_10_0_1000_1_1, tree_near_10_0_1000_2_1, tree_near_10_0_1000_3_1	tree_far_10_0_1000_1_1, tree_far_10_0_1000_2_1, tree_far_10_0_1000_3_1

Testtabelle	test_db_10_0_100, …, test_db_10_0_5000	test_db_10_0_100, …, test_db_10_0_5000	test_db_10_0_100, …, test_db_10_0_5000
Input-Datei	input_db_10_0	input_near_10_0	input_far_10_0
Output-Datei	output_db_10_0	output_near_10_0	output_far_10_0

Tabelle 25: Parameter der Testreihen des Entscheidungsbaum-Klassifikators

Der DT-Algorithmus liefert als Ergebnis einen Klassifikator in Form eines Entscheidungsbaumes, der in einer Baumtabelle in der Datenbank abgelegt wird. Zusätzlich dazu wird für jeden Testlauf eine Ergebnistabelle erstellt, welche die Testdaten enthält und um eine Dimension „PREDICTED_CLASS“ für die vom Klassifikator zugewiesene Klasse erweitert wurde. Außerdem werden Kennzahlen (Qualitätskriterien) bzgl. der Qualität und Güte des Klassifikators (vgl. Kapitel 3) berechnet.

Baumtabellen

Jeder erzeugte Entscheidungsbaum wird in der Datenbank wie in Kapitel 7.2.3 erläutert abgelegt und erhält eine eindeutige Bezeichnung folgender Form:

TREE_XXXX_Y1_Y2_ZZZZ_A_I

TREE... Kennzeichen einer Baumtabelle
XXXX... Ursprung der Trainingsdaten - Zufallspunkte (DB), nahe Punkte (NEAR), entfernte Punkte (FAR)
Y1… Anzahl der numerischen Attribute der zugehörigen Trainingstabelle
Y2... Anzahl kategorischen Attribute der zugehörigen Trainingstabelle
ZZZZ... Anzahl der Tupel der Trainingsdaten
A... Splitstrategie - 0hne Verteilung (1), Min/Max-Strategie (2), Dichtefunktion-Strategie (3)
I... laufende Nummer

Der Name der Baumtabelle fließt beim Start des Algorithmus als Parameter in die Entscheidungsbaum-Konstruktion ein.

Ergebnistabellen

Für jeden Testlauf über einen Entscheidungsbaum wird eine Ergebnistabelle erstellt. Sie hat die Struktur der Testtabelle und besitzt eine zusätzliche Spalte für die vom DT-Algorithmus ermittelte Klasse (PREDICTED_CLASS). Jede Ergebnistabelle erhält eine eindeutige Bezeichnung folgender Form:

TREE_XXXX_Y1_Y2_ZZZZ_A_I_R_J

TREE... Kennzeichen einer Baumtabelle (Jedes Testergebnis bezieht sich auf einen Baum)
XXXX... Ursprung der Trainingsdaten - Zufallspunkte (DB), nahe Punkte (NEAR), entfernte Punkte (FAR)
Y1… Anzahl der numerischen Attribute der zugehörigen Trainingstabelle
Y2... Anzahl kategorischen Attribute der zugehörigen Trainingstabelle
ZZZZ... Anzahl der Tupel der Trainingsdaten
A... Splitstrategie - 0hne Verteilung (1), Min/Max-Strategie (2), Dichtefunktion-Strategie (3)
I... laufende Nummer
R... Ergebnistabelle
J... Nummer des Tests auf aktuellen Entscheidungsbaum-Klassifikator

Der Name der Ergebnistabelle wird zur Laufzeit durch den DT-Algorithmus aus der Testtabelle des aktuellen Testlaufs generiert.

Kennzahlen

Die Kennzahlen eines Entscheidungsbaumes sind Kriterienwerte, die Aussagen über die Qualität und Güte eines Klassifikators treffen. Nach der Konstruktion eines Entscheidungsbaums werden als Kennzahlen der Kompaktheit die

- Anzahl der Knoten des Baumes und die
- Höhe des Baumes

geliefert, die für alle Testläufe einer Testreihe gleich sind. Jeder Testlauf mit einer Testtabelle liefert zusätzlich folgende Informationen über einen Entscheidungsbaum:

- Anzahl der Testtupel

- Anzahl der richtig klassifizierten Tupel
- Anzahl der falsch klassifizierten Tupel
- Präzision (precision)
- Vollständigkeit (recall)
- Genauigkeit (accuracy)
- Klassifikationsgenauigkeit
- Tatsächlicher Klassifikationsfehler
- Beobachteter Klassifikationsfehler

Der beobachtete Klassifikationsfehler besitzt nur bei Tests über die Trainingsmenge selbst einen gültigen Wert. Die Werte für precision, recall und accuracy werden bei einem Test über die Trainingsmenge nicht in die Bewertung miteinbezogen.

Nachfolgend werden die Ergebnisse der Standardtestreihen mit fünf numerischen/fünf kategorischen und zehn numerischen Attributen in den Trainingsdaten in tabellarischer und grafisch aufbereiteter Form dargestellt. Wurde in einem Testlauf eine Kennzahl (Qualitätskriterium) nicht berechnet oder liefert einen ungültigen Wert, so besitz sie den Wert -1.
Im Anschluss an die Standardtestreihen werden jene Tests diskutiert, bei denen sich unter der Verwendung von Hilfsinformationen Qualitätsunterschiede zu den Standardtests ergeben haben. Die Gesamtheit aller durchgeführten Testreihen ist dem Anhang zu entnehmen.

8.2.1 Standardtestreihen

Die Standardtestreihen verwenden bei der Konstruktion der Entscheidungsbäume keinerlei Art von Hilfsinformationen aus dem ersten Schritt des kombinierten Prozesses (K-Means). Die Trainingsdaten bestehen aus zufälligen Punkten der Ausgangsdaten. Im Standardfall werden zwei Entscheidungsbäume erzeugt. Die Standardtestreihen 1 und 2 entsprechen den Testreihen 1 und 4 ohne Hilfsinformationen (vgl. Kapitel 8.2).

Standardtestreihe 1: 5 numerische Dimensionen, 5 kategorische Dimensionen und zufällige Punkte als Trainingsdaten

Diese Testreihe verwendet einen Entscheidungsbaum, dessen Trainingsmenge aus fünf numerische und fünf kategorische Dimensionen besteht. Als Splitstrategie wurde die Strategie ohne Verteilungsinformation verwendet. Die Tabelle 25 zeigt die Testergebnisse über den konstruierten Baum in Form von Werten der Qualitätskriterien.

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	23								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	71	134	208	350	702	1410	2127	3538	711
falsch klassifizierte Tupel	29	66	92	150	298	590	873	1462	289
Präzision (precision)	0,31	0,36	0,35	0,41	0,42	0,42	0,43	0,45	-1
Vollständigkeit (recall)	0,76	0,80	0,82	0,83	0,83	0,83	0,83	0,84	-1
Genauigkeit (accuracy)	0,44	0,51	0,5	0,52	0,53	0,53	0,54	0,55	-1
Klassifikationsgenauigkeit	0,71	0,67	0,69	0,70	0,70	0,71	0,71	0,71	0,71
Tatsächlicher Klassifikationsfehler	0,29	0,33	0,31	0,30	0,30	0,29	0,29	0,29	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,29

Tabelle 26: Testergebnisse der Standardtestreihe 1

Standardtestreihe 2: 10 numerische Dimensionen und zufällige Punkte als Trainingsdaten

Diese Testreihe verwendet einen Entscheidungsbaum, dessen Trainingsmenge aus zehn numerischen Attributen besteht. Die Splitpunkte wurden mittels der Strategie ohne Verteilungsinformation bestimmt. Die Tabelle 26 enthält die Werte der Qualitätskriterien des konstruierten Baums für alle Testläufe.

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	9								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	74	139	209	350	721	1420	2154	3561	723
falsch klassifizierte Tupel	26	61	91	150	279	580	846	1439	277
Präzision (precision)	0,32	0,36	0,32	0,33	0,34	0,33	0,34	0,35	-1
Vollständigkeit (recall)	1,00	0,98	0,99	0,98	0,99	0,99	0,99	0,99	-1
Genauigkeit (accuracy)	0,50	0,55	0,52	0,53	0,53	0,53	0,52	0,53	-1
Klassifikationsgenauigkeit	0,74	0,70	0,70	0,70	0,72	0,71	0,72	0,71	0,72
Tatsächlicher Klassifikationsfehler	0,26	0,30	0,30	0,30	0,28	0,29	0,28	0,29	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,28

Tabelle 27: Testergebnisse der Standardtestreihe 2

Unter dem Gesichtspunkt der Klassifikationsgenauigkeit und der accuracy verhalten sich beide Entscheidungsbäume im Standardfall sehr ähnlich. Abweichungen ergeben sich in den Werten für Präzision und Vollständigkeit. Bei der Verwendung einer Trainingsmenge, die ausschließlich numerische Dimensionen enthält, liegt der Wert für die Vollständigkeit annähernd bei 1,0. In Standardtestreihe 1 schwankt der Wert um 0,8. Im Gegensatz dazu verhalten sich die Werte für die Präzision entgegengesetzt. In Testreihe 1 liegt der Wert für die Präzision um durchschnittlich 5-10% höher als in Testreihe 2. Daraus lässt sich schlussfolgern, dass bei der Erzeugung von Entscheidungsbäumen aus numerischen Attributen die Wahrscheinlichkeit größer ist, dass relevante Datensätze richtig klassifiziert werden. Andererseits kann angenommen werden, dass bei der Verwendung einer Trainingsmenge mit fünf numerischen und 5 kategorischen Dimensionen die Wahrscheinlichkeit größer ist, dass richtig klassifizierte Datensätze auch relevant sind. Eine weitere Interpretation, die keinesfalls außer Acht gelassen werden darf, ist die Tatsache, dass das vorgelagerte Clusteringverfahren „K-Means“ nur numerische Dimension in den Clusteringprozess miteinbezieht. Die kategorischen Dimensionen bestimmen die Clusterzugehörigkeiten nicht. Somit ist die Chance bei der Verwendung von kategorischen Dimensionen in den Trainingdaten groß, dass den

Testdatensätzen Klassen zugewiesen werden, die von den Clustern abweichen. Abbildung 41 zeigt den Qualitätsunterschied der beiden Bäume aus den Standardtestreihen anhand der Klassifikationsgenauigkeit.

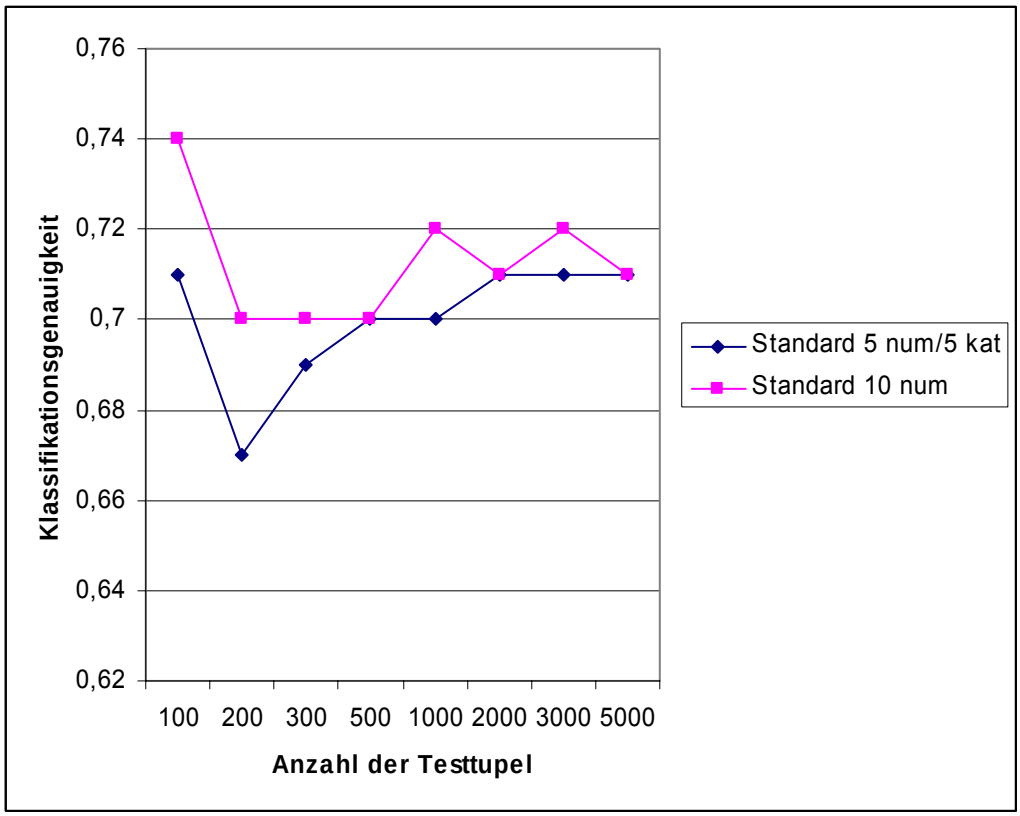


Abbildung 41: Klassifikationsgenauigkeit der Standardtestreihen

Des Weiteren ergibt sich ein großer Unterschied in der Anzahl der Knoten der Bäume. Während der Baum in Standardtestreihe 1 aus 23 Knoten besteht, besitzt der Baum in Standardtestreihe 2 lediglich 9 Knoten. Dies ist darauf zurückzuführen, dass numerische Attribute binär gesplittet werden, kategorische Attribute jedoch für jede Attributausprägung einen Sohnknoten im Baum einnehmen. Die kategorischen Attribute in diesen Tests beinhalten bis zu maximal sechs verschiedene Attributausprägungen. Abbildung 42 visualisiert den Entscheidungsbaum aus Standardtestreihe 2 (zehn numerische Dimensionen). Dieser Baum wird in späterer Folge dazu herangezogen, Unterschiede in der Kompaktheit im Vergleich zu anderen Entscheidungsbäumen zu demonstrieren.

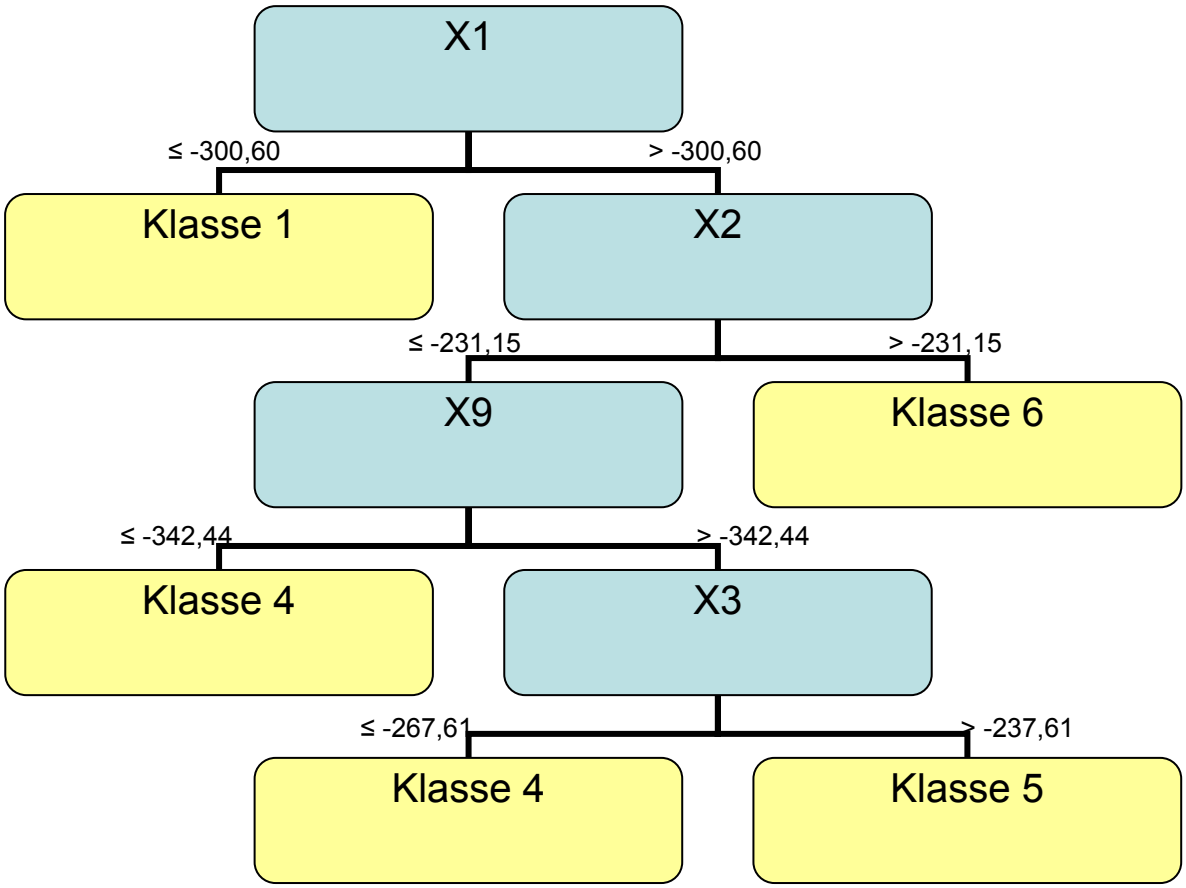


Abbildung 42: Entscheidungsbaum aus zehn numerischen Dimensionen im Standardfall

8.2.2 Testreihen mit Hilfsinformationen

Für die Diskussion der Testreihen mit Hilfsinformationen werden jene Entscheidungsbäume und Testergebnisse gezeigt, die im Vergleich zu den Standardtestreihen den größten positiven oder negativen Einfluss auf die Qualität der Klassifikatoren verdeutlichen. Die Testergebnisse über Entscheidungsbäume, die unter Verwendung von Hilfsinformationen bzw. von Kombinationen von Hilfsinformationen erzeugt wurden, werden nachfolgend in tabellarischer Form dargestellt. Für die Veranschaulichung der Kompaktheit eines Modells werden aus Gründen der Übersichtlichkeit ausschließlich Entscheidungsbäume visualisiert, die aus einer Trainingsmenge mit zehn numerischen Attributen konstruiert wurden. Alle übrigen Tests sind den Testreihen im Anhang zu entnehmen.

Einsatz von Splitstrategien auf zufällige Trainingsdaten

Der Einsatz von Splitstrategien mit Hilfsinformationen bei einer zufällig verteilten Trainingsmenge bringt einen geringen bis mittleren Qualitätszuwachs im Sinne der Klassifikationsgenauigkeit. Die Min/Max-Strategie liefert für eine Trainingsmenge mit fünf numerischen und fünf kategorischen Dimensionen sogar schlechtere Ergebnisse als die Splitstrategie ohne Verteilungsinformation. Dafür besitzen Präzision (precision) und Genauigkeit (accuracy) erhöhte Werte (vgl. Tabelle 28). Die Minima und Maxima als potentielle Splitpunkte führen somit dazu, dass mehr relevante Datensätze in den richtig klassifizierten Tupeln enthalten sind.

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	21								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	67	128	191	320	651	1298	1960	3269	654
falsch klassifizierte Tupel	33	72	109	180	349	702	1040	1731	346
Präzision	0,33	0,38	0,38	0,44	0,45	0,46	0,47	0,49	-1
Vollständigkeit	0,76	0,80	0,82	0,83	0,83	0,83	0,83	0,84	-1
Genauigkeit	0,48	0,54	0,55	0,58	0,58	0,59	0,59	0,60	-1
Klassifikationsgenauigkeit	0,67	0,64	0,64	0,64	0,65	0,65	0,65	0,65	0,65
Tatsächlicher Klassifikationsfehler	0,33	0,36	0,36	0,36	0,35	0,35	0,35	0,35	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,35

Tabelle 28: Testergebnisse - Zufällige Trainingsdaten und Min-/Max-Strategie

Dagegen liefert die Dichtefunktion-Strategie im Vergleich zum Verfahren ohne Verteilungsinformation eine leicht erhöhte Klassifikationsgenauigkeit. Der erzeugte Baum enthält darüber hinaus weniger Knoten und ist weniger hoch. Somit ist dieser Baum kompakter als der Baum der Standardtestreihe (vgl. Tabelle 26) und daher qualitativ hochwertiger. Die Tabelle 29 enthält die Testergebnisse auf einen Entscheidungsbaum, der mithilfe der Dichtefunktion-Strategie und zufälligen Trainingsdaten erzeugt wurde. Die Trainingsdaten enthalten fünf numerische und

fünf kategorische Dimensionen. Jeglicher Qualitätszuwachs im Vergleich zur Standardtestreihe ist farbig markiert.

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	18								
Höhe des Baumes	4								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	73	137	214	359	715	1439	2167	3596	718
falsch klassifizierte Tupel	27	63	86	141	285	561	833	1404	282
Präzision	0,29	0,33	0,32	0,38	0,39	0,39	0,40	0,42	-1
Vollständigkeit	0,72	0,75	0,78	0,78	0,78	0,79	0,79	0,79	-1
Genauigkeit	0,40	0,47	0,45	0,48	0,49	0,49	0,49	0,50	-1
Klassifikationsgenauigkeit	0,73	0,69	0,71	0,72	0,72	0,72	0,72	0,72	0,72
Tatsächlicher Klassifikationsfehler	0,27	0,31	0,29	0,28	0,28	0,28	0,28	0,28	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,28

Tabelle 29: Testergebnisse - Zufällige Trainingsdaten und Dichtefunktion-Strategie

Für eine Trainingsmenge mit zehn numerischen Dimensionen hat sich gezeigt, dass die Dichtefunktion-Strategie ebenfalls zu einem kompakteren Modell führt. Das bedeutet, dass diese Strategie in jedem Fall zu einem kompakteren Modell führt. Die erzeugte Kompaktheit geht aber in einigen Fällen auf Kosten der Klassifikationsgenauigkeit. Der in Abbildung 40 ersichtliche Baum ist durch Anwendung der Dichtefunktion-Strategie konstruiert worden. Im Vergleich zum Baum aus Abbildung 43 fällt im Baum unten die Klasse 4 weg. Übrig bleiben jene drei Klassen, die rund 65% der Trainingsdaten ausmachen. Die Trainingsdaten bestehen aus zufälligen Punkte mit zehn numerischen Dimensionen (vgl. Tabelle 10_db_3 im Anhang).

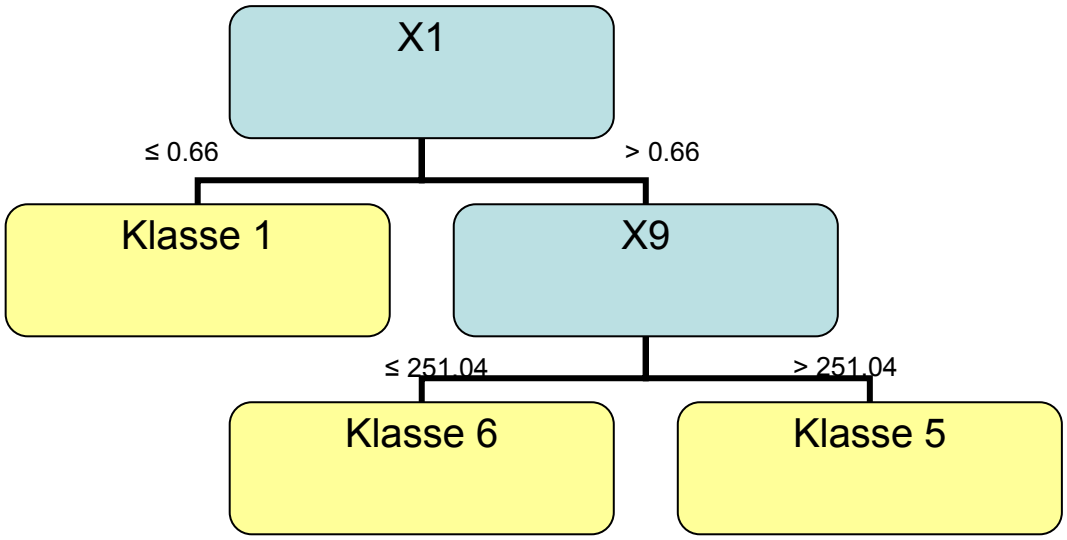


Abbildung 43: Entscheidungsbaum aus zehn numerischen Attributen bei Dichtefunktion-Strategie

Dagegen erhält man für die Min/Max-Strategie eine erhebliche Qualitätssteigerung hinsichtlich der Klassifikationsgenauigkeit. Diese Strategie erzeugt zwar ein weniger kompaktes Modell, führt aber zu einem Sinken des tatsächlichen Klassifikationsfehlers um mehr als 10% (vgl. Tabelle 10_db_2 im Anhang).

Abbildung 44 zeigt noch einmal grafisch den Qualitätsunterschied bei der Verwendung unterschiedlicher Splitstrategien auf zufällige Trainingsdaten. Das Diagramm stellt jene Testergebnisse dar, die im Vergleich zu den Standardtestreihen eine erhöhte Klassifikationsgenauigkeit aufweisen. Die Ergebnisse stammen einerseits aus einer Testreihe mit fünf numerischen und fünf kategorischen Dimensionen unter Anwendung der Dichtefunktion-Strategie (vgl. Tabelle 55_db_3 im Anhang), andererseits aus einer Testreihe mit zehn numerischen Dimensionen und Verwendung der Min/Max-Strategie (vgl. Tabelle 10_db_2 im Anhang).

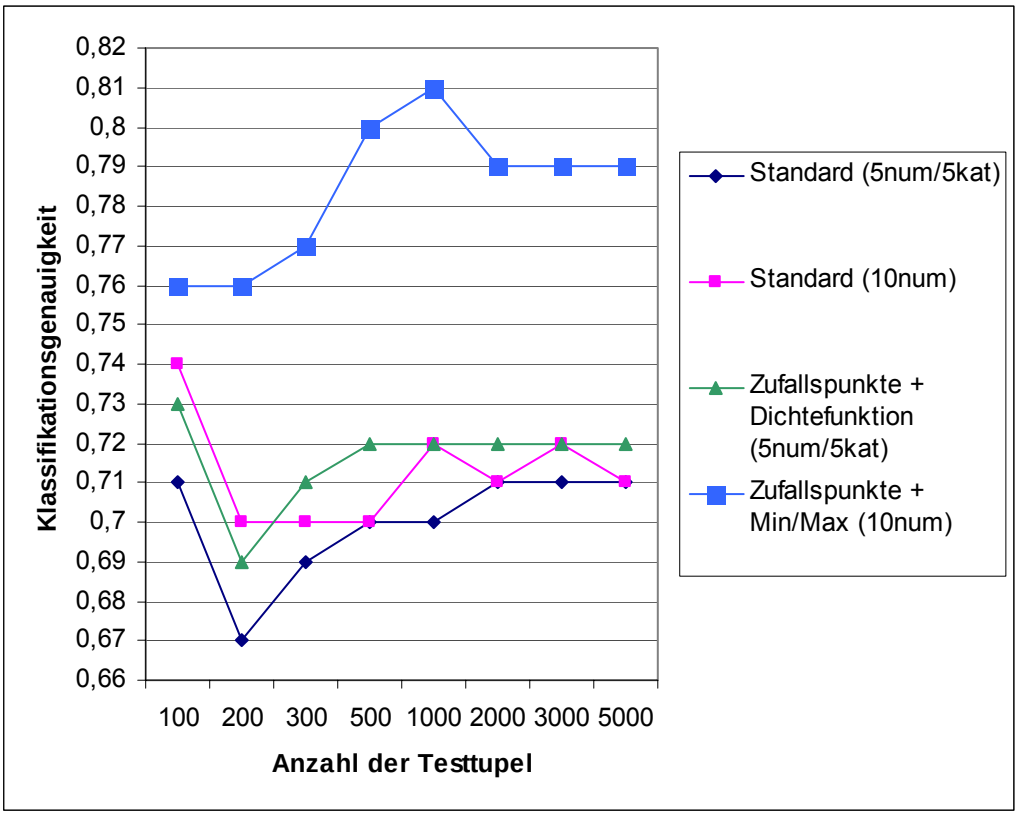


Abbildung 44: Klassifikationsgenauigkeit – Splitstrategien mit Hilfsinformation und zufällige Punkte

Einsatz von Trainingsdaten mit Lageeigenschaften

Die Verwendung von den Clustermittelpunkten nahen Punkten als Trainingsdaten führt in jedem Fall zu einer Qualitätssteigerung der Klassifikatoren. Der Einsatz von nahen Punkten in Verbindung mit der Splitstrategie ohne Verteilungsinformation bringt einen erheblichen Anstieg der Klassifikationsgenauigkeit unabhängig davon, wie die Trainingsdaten zusammengesetzt sind (nur numerische oder numerische und kategorische Dimensionen). Der geringe Verlust von Präzision und Genauigkeit im Sinne der Relevanz ist vernachlässigbar. Der große Qualitätsanstieg ist darauf zurückzuführen, dass die eng beieinander liegenden Cluster der Ausgangsdaten durch die Verwendung von nahen Punkten besser voneinander getrennt wurden. Solche Cluster sind in Abbildung 40 in Kapitel 8.1 rechts unten zu erkennen. Ein sehr dichter Cluster in der Mitte wird von zwei eher losen Clustern umschlossen.
Testdaten, die im Randbereich zwischen zwei Clustern liegen, werden eher richtig zugeordnet als dies bei zufälligen Trainingsdaten der Fall ist. Abbildung 45 zeigt die Klassifikationsgenauigkeit bei der Verwendung von nahen Punkten im Vergleich zu zufälligen Punkten in den Trainingsdaten.

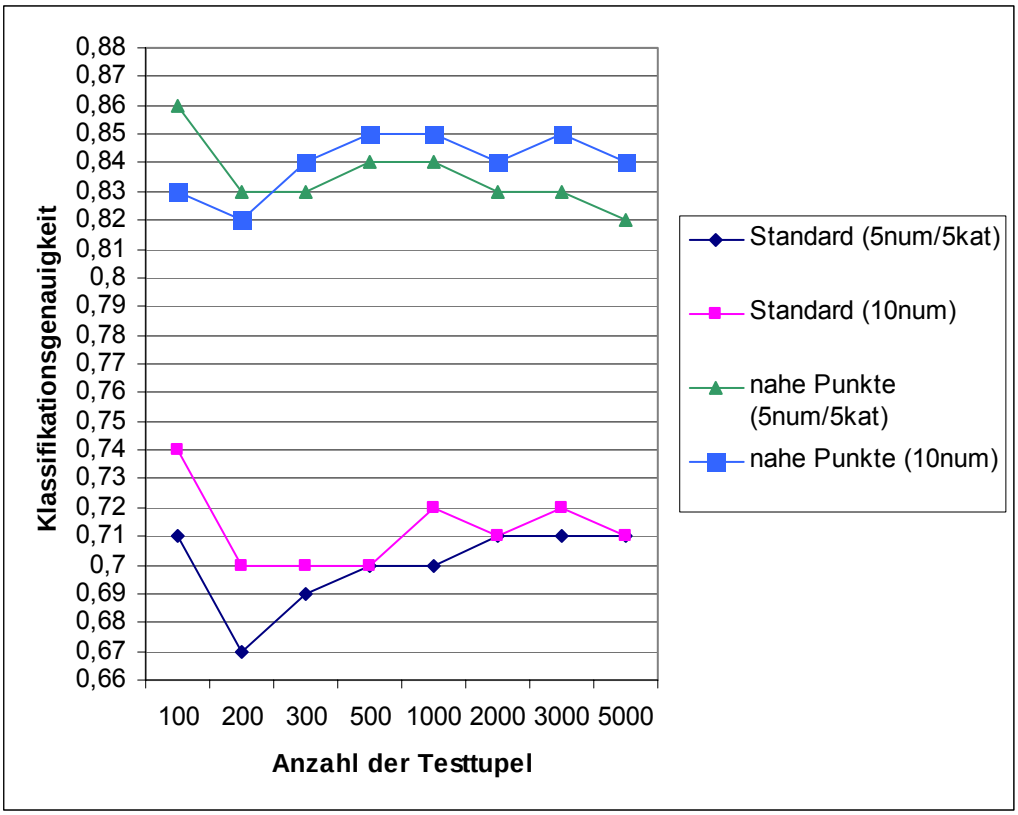


Abbildung 45: Klassifikationsgenauigkeit – nahe Punkte

Werden den Clustermittelpunkten weit entfernte Punkte als Trainingsdaten eingesetzt, so konnte nur bei der Verwendung von zehn numerischen Dimensionen eine Steigerung der Klassifikationsgenauigkeit festgestellt werden. Dies ist darauf zurückzuführen, dass die Verschiebung der Trainingsdaten nach weit außen bei der Verwendung von kategorischen Dimensionen eine große Veränderung der Attributausprägungen mit sich bringen kann. Der Unterschied dieser Verschiebung wirkt sich bei kategorischen Dimensionen schwerwiegender aus, da sie im Normalfall wenige diskrete Ausprägungen beinhalten. Die Abweichung der außen liegenden Punkte in den kategorischen Dimensionen im Vergleich zum Clustermittelpunkt ist verhältnismäßig größer als bei numerischen (stetigen) Dimensionen. In Abbildung 46 ist die Klassifikationsgenauigkeit bei der Verwendung von weit entfernten Punkten dargestellt.

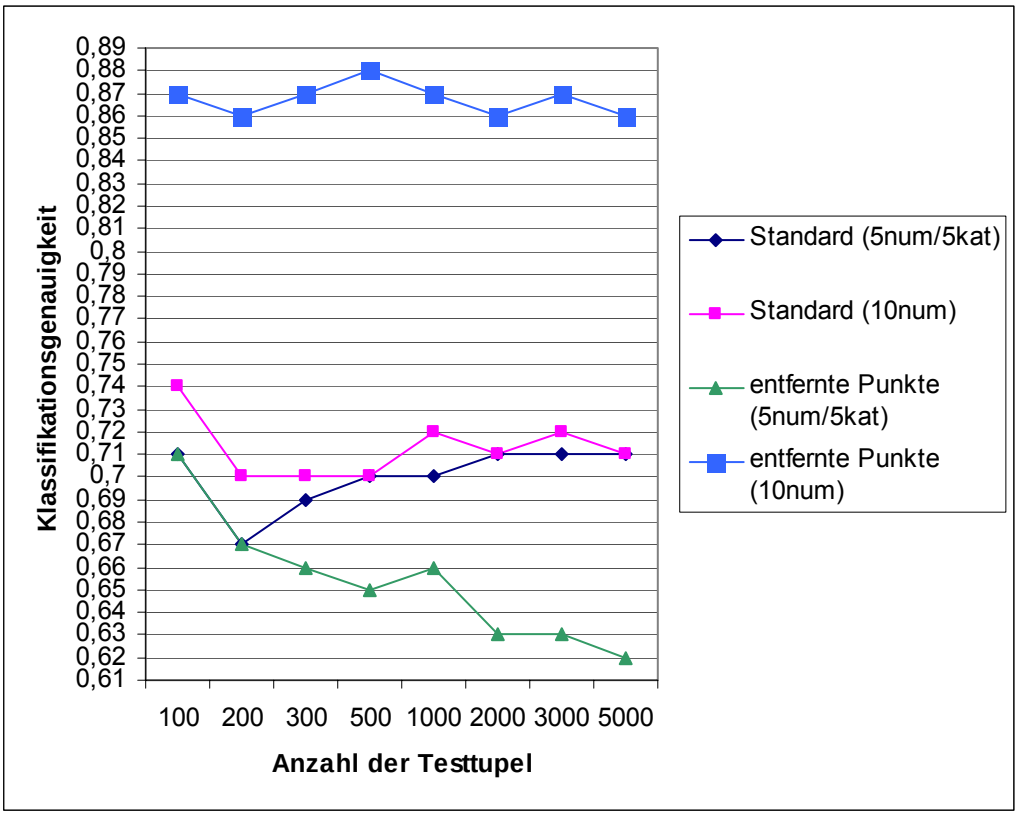


Abbildung 46: Klassifikationsgenauigkeit – entfernte Punkte

Einsatz von Kombinationen der Hilfsinformationen

Die Kombination von Verteilungsinformationen und Trainingsdaten mit Lageeigenschaften als Hilfsinformation bringt in der Hälfte der Fälle eine Qualitätssteigerung im Vergleich zu den Standardtestreihen. Die Kombination von entfernten Punkten als Trainingsmenge mit Verteilungsinformationen führt in keiner Testreihe zu einem positiven Qualitätsunterschied. Besonders die Kombination von entfernten Punkten mit der Min/Max-Strategie erzeugt eine drastische Erhöhung des tatsächlichen Klassifikationsfehlers. In Tabelle 30 sind die Testergebnisse über einen Baum dieser Hilfsinformations-Kombination dargestellt. Die Klassifikationsgenauigkeit liegt unter 50% und das Modell ist im Vergleich zur Standardtestreihe wenig kompakt.

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	27								
Höhe des Baumes	11								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1048
richtig klassifizierte Tupel	43	85	133	220	424	849	1270	2080	546
falsch klassifizierte Tupel	57	115	167	280	576	1151	1730	2920	502
Präzision	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-1
Vollständigkeit	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-1
Genauigkeit	0,33	0,32	0,33	0,32	0,32	0,34	0,33	0,33	-1
Klassifikationsgenauigkeit	0,43	0,43	0,44	0,44	0,42	0,42	0,42	0,42	0,52
Tatsächlicher Klassifikationsfehler	0,57	0,57	0,56	0,56	0,58	0,58	0,58	0,58	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,48

Tabelle 30: Testergebnisse – Entfernte Trainingsdaten-Punkte und Min/Max-Strategie

Der Einsatz von Trainingsdaten mit Lageeigenschaften alleine ist in den meisten Fällen der kombinierten Anwendung vorzuziehen. Die Kombination bringt keinen Qualitätsgewinn sondern oftmals einen Qualitätsverlust des Klassifikators.

Trotzdem ist eine Kombination der Hilfsinformationen nicht völlig sinnlos. Die gemeinsame Anwendung von nahen Punkten und der Dichtefunktion-Strategie erzeugt einen Entscheidungsbaum, der annähernd die gleiche Klassifikationsgenauigkeit wie ein Baum, der alleine aus nahen Punkten konstruiert wurde, besitzt. Die Dichtefunktion-Strategie erzeugt außerdem Bäume, die zum Großteil kompaktere Modelle repräsentieren als das ohne die Verwendung von Hilfsinformation der Fall ist. Abbildung 47 zeigt die Klassifikationsgenauigkeit von Bäumen aus nahen Punkten und Bäumen, die aus der Kombination von nahen Punkten und der Dichtefunktion-Strategie entwickelt wurden.

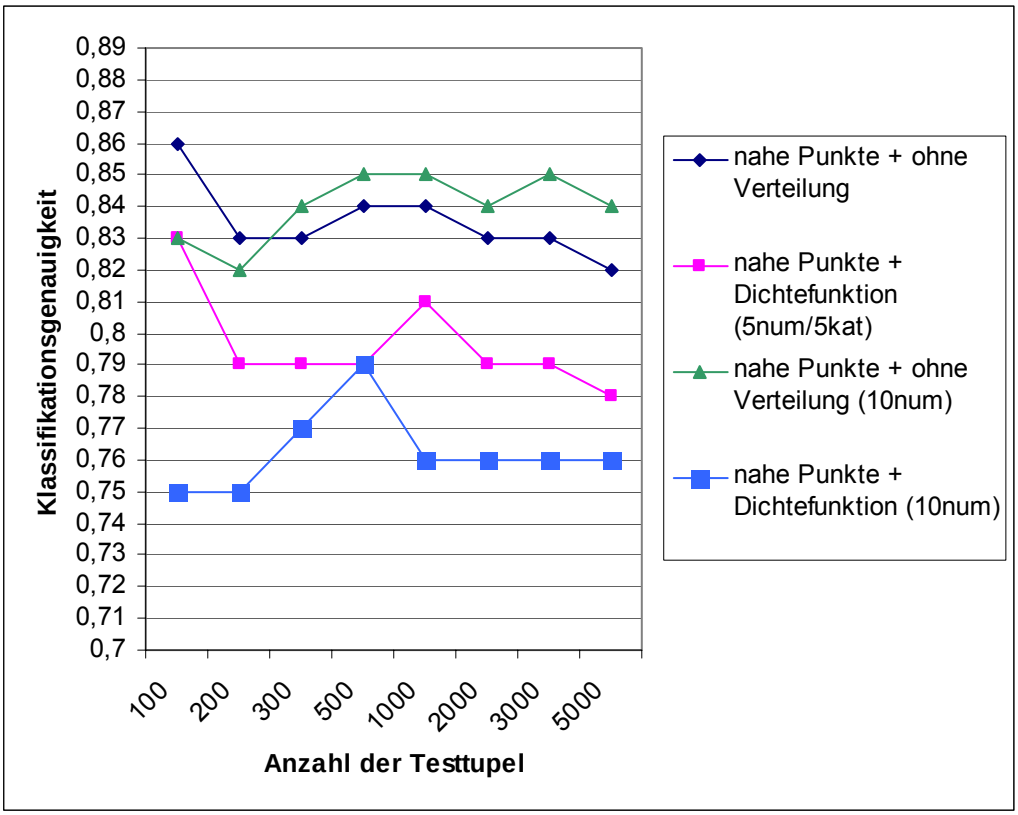


Abbildung 47: Klassifikationsgenauigkeit – nahe Punkte und nahe Punkte mit Dichtefunktion-Strategie

Trotzdem liegt der Vorteil der Kombination nicht unbedingt in der Steigerung der Qualität des Klassifikators, sondern in der Effizienz der Generierung des Entscheidungsbaumes (vgl. Kapitel 7.2.2). Auch wenn eine Effizienzsteigerung kein vorrangiges Ziel dieser Arbeit ist, muss erwähnt werden, dass die Kombination dieser Hilfsinformationen schneller zu einem qualitativ vergleichbaren Ergebnis führt. Tabelle 31 enthält beispielhaft für eine zufällige Trainingsmenge mit zehn numerischen Dimensionen die Laufzeiten der Klassifikator-Konstruktion für die drei verwendeten Splitstrategien.

Splitstrategie	Baumknoten	Laufzeit (in sec)	Laufzeit pro Knoten (sec/Knoten)
Ohne Verteilung	9	87	9,67
Min-/Max-Strategie	13	70	5,38
Dichtefunktion-Strategie	5	36	7,20

Tabelle 31: Laufzeit für die Klassifikator-Konstruktion

Im nun folgenden und abschließenden Kapitel werden alle verwendeten Hilfsinformationen einander gegenübergestellt und hinsichtlich ihrer Brauchbarkeit für die Konstruktion von Entscheidungsbäumen bewertet.

8.3 Brauchbarkeit der Hilfsinformationen

Die in [SK04] identifizierten Hilfsinformationen wurden in dieser Arbeit nicht in ihrer Gesamtheit verwendet. Es sind nur solche Hilfsinformationen in die Tests eingeflossen, die als „potentiell verwertbar“ (vgl. Kapitel 6.2) bei der Konstruktion von Entscheidungsbäumen erachtet wurden. Um die Gruppe der „potentiell verwertbaren Hilfsinformationen“ zu bewerten, wird der Begriff der Brauchbarkeit eingeführt. Als brauchbar werden hier alle Hilfsinformationen bezeichnet, die

- **potentiell verwertbar** sind und einen
- **positiven Einfluss auf die Qualität** des Klassifikators

haben. Darüber hinaus werden auch solche Hilfsinformationen als brauchbar eingestuft, die die Konstruktion des Klassifikators unterstützen und nicht zum Qualitätsverlust führen.

Die Brauchbarkeit wird im Folgenden mit einer Skala von 1 bis 5 bewertet, wobei 1 für „sehr brauchbar“, 2 für „gut brauchbar“, 3 für „brauchbar“, 4 für „bedingt brauchbar“ und 5 für „unbrauchbar“ steht. Die Bewertung erfolgt über die Tests aller Testreihen unabhängig davon, ob die Dimensionen numerisch oder kategorisch sind. Die verwerteten Hilfsinformationen werden teilweise gemeinsam bewertet, d.h. dass z.B. das Minimum und das Maximum eines Clusters zusammen bewertet werden, da sie nur kombiniert als Hilfsinformationen in die Konstruktion der Entscheidungsbäume eingeflossen sind.

Die Tabelle 30 enthält alle getesteten Hilfsinformationen und Kombinationen von Hilfsinformationen mit ihrer dazugehörigen Bewertung der Brauchbarkeit.

<table>
<tr><th rowspan="2">Hilfsinformationen</th><th colspan="3">Trainingsdaten</th></tr>
<tr><th>Zufällige Punkte</th><th>Nahe Punkte</th><th>Entfernte Punkte</th></tr>
<tr><td>Keine Hilfsinformation</td><td></td><td>1</td><td>4</td></tr>
<tr><td>Minimum des Clusters</td><td rowspan="2">3</td><td rowspan="2">2</td><td rowspan="2">5</td></tr>
<tr><td>Maximum des Clusters</td></tr>
<tr><td>Mittelwert des Clusters</td><td rowspan="2">4</td><td rowspan="2">2</td><td rowspan="2">3</td></tr>
<tr><td>Standardabweichung des Clusters</td></tr>
</table>

Tabelle 32: Bewertung der Hilfsinformationen

Die Brauchbarkeit der Hilfsinformationen ist mit großer Wahrscheinlichkeit stark von der initialen Verteilung der Ausgangsdaten (vgl. Abbildung 40 in Kapitel 8.1) und somit implizit von der Verteilung der Trainingsdaten abhängig. Die besten Ergebnisse für die vorliegenden Trainingsdaten liefern der Einsatz von nahen Punkten als Trainingsdaten bzw. deren Kombination mit der Dichtefunktion-Strategie. Nahe Punkte trennen die Cluster besser von einander. Die Dichtefunktion-Strategie liefert in alle Tests ein mindestens gleich kompaktes oder kompakteres Modell im Vergleich zu den Ergebnissen der Standardtestreihen. Im Unterschied dazu liefert die Kombination von entfernten Punkten und der Min/Max-Strategie in keinem Testlauf ein besseres Ergebnis. Die Klassifikationsgenauigkeit der so erzeugten Bäume ist in jedem Fall um vieles niedriger als jene der Bäume, die durch die Standardtestreihen erzeugt worden sind. Um die hier durchgeführten Bewertungen nachvollziehen zu können, sind alle durchgeführten Test im Anhang ersichtlich.

9 Fazit

Diese Arbeit hat den in [SK04] erstmals definierten Begriff des „Kombinierten Data Mining“ aufgegriffen und erweitert. Das „Kombinierten Data Mining“ wurde in die Typen naiv, Vorgänger kennt Nachfolger und Nachfolger kenn Vorgänger unterteilt. Zusätzlich zu diesen drei Typen wurde an dieser Stelle das „implizite Kombinierte Data Mining“ eingeführt, bei dem ein Data-Mining-Verfahren die Techniken eines anderen Verfahrens zur Lösung einer Aufgabenstellung verwendet. Existierende Ansätze und Anwendungen realisieren lediglich naives „Kombiniertes Data Mining“ (vgl. Kapitel 4.1).

In [SK04] wurden zwei Clustering-Algorithmen entwickelt, die den Typ „Vorgänger kennt Nachfolger“ realisieren. Ziel dieser Arbeit war es, einen Klassifikator zu implementieren, der es ermöglicht, Hilfsinformationen aus einem vorher ausgeführten Verfahren anzuwenden. Der Klassifikator realisiert den Typ „Nachfolger kennt Vorgänger“ und nutzt verwertbare Hilfsinformationen zur Erreichung einer Qualitätssteigerung. Die Qualität des Klassifikators wurde mittels mehrerer Kennzahlen bzw. Qualitätskriterien (siehe Kapitel 3.2) gemessen. Die wichtigste Kennzahl bei der Klassifikation stellt die Klassifikationsgenauigkeit dar. Zur Bestimmung der Kennzahlen wurde ein „Entscheidungsbaum-Klassifikator“ entwickelt, der die Hilfsinformationen aus dem vorgelagerten Clusteringverfahren „K-Means“ verwerten kann. Darüber hinaus wurde bestimmt, welchen Einfluss die untersuchten Hilfsinformationen auf die Klassifikatorqualität ausüben, d.h. es wurde ermittelt, welche Hilfsinformationen sich besonders gut für die Klassifikation eignen.
Zu diesem Zweck wurden die identifizierten Hilfsinformationen in Kategorien eingeteilt (vgl. Kapitel 6.1). Bei der Implementierung des Klassifikators wurden nur solche Hilfsinformationen berücksichtigt, die als „potentiell verwertbar“ erachtet wurden (vgl. Kapitel 6.2).

Die Qualitätstests umfassten die Miteinbeziehung der Hilfsinformationen „Lageeigenschaften der Trainingsdaten“ und „Verteilungsinformationen zu den Clustern“ bzw. ihre Kombination. Trainingsdaten mit Lageeigenschaften waren in Form von separaten Ergebnistabellen aus dem ersten Schritt des „Kombinierten Data

Mining“ vorhanden. Aus den in „K-Means“ berechneten Verteilungsinformationen wurden die Splitstrategien „Min-/Max-Strategie“ und „Dichtefunktion-Strategie“ entwickelt. Zur Qualitätsbestimmung wurden sechs Testreihen mit jeweils unterschiedlicher Kombination von Hilfsinformationen durchgeführt. In jeder Testreihe wurden drei Klassifikatoren erstellt und die dazugehörigen Qualitätskriterien berechnet.

Für die verwendeten Trainingsdaten stellte sich heraus, dass die alleinige Verwendung von „nahen Punkten“ die Klassifikationsgenauigkeit um mindestens 10% steigert. Die Kombination von „nahen Punkten“ mit „Verteilungsinformationen zu den Clustern“ bringt eine etwas geringere Qualitätsverbesserung, liefert jedoch in den meisten Fällen ein kompakteres Modell des Klassifikators in geringerer Laufzeit (vgl. Kapitel 8.2.2. Einsatz von Kombinationen der Hilfsinformationen).
Bei der Verwendung von „entfernten Punkten“ als Trainingsdaten stellte sich lediglich dann eine Qualitätssteigerung ein, wenn ausschließlich numerische Dimensionen zur Klassifikation herangezogen wurden. Die Steigerung der Klassifikationsgenauigkeit erreichte dabei aber eine Erhöhung von durchschnittlich 15%. In der Kombination mit Verteilungsinformationen wurde in Verbindung mit der „Dichtefunktion-Strategie“ ein annähernd gleiches Ergebnis wie bei der Durchführung ohne Hilfsinformationen erreicht.
Die Anwendung von Splitstrategien, die mittels Hilfsinformationen erzeugt wurden, auf eine zufällige Trainingsmenge muss als „bedingt brauchbar“ bewertet werden. Während „Minimum und Maximum“ bei einer rein numerischen Trainingsmenge eine Qualitätssteigerung bewirkten, führte der Einsatz von „Mittelwert und Standardabweichung“ bei Trainingsdaten mit sowohl numerischen als auch kategorischen Dimensionen zu erhöhter Qualität. Auch wenn die Splitstrategien auf kategorische Dimensionen keinen Einfluss ausüben, ist die Qualitätssteigerung darauf zurückzuführen, dass die ausgewählten numerischen Dimensionen die Cluster der Traingsmenge besonders gut repräsentieren.

Quellenverzeichnis

[ALVA] Sergio A. Alvarez: „An exact analytical relation among recall, precision, and classification accuracy in information retrieval", Department of Computer Science, Boston College

[BH03] Helmut Beran: "Skriptum Statistik für Wirtschaftsinformatiker", Abteilung für Angewandte Systemforschung und Statistik, Johannes Kepler Universität Linz, 2003

[BL97] Berry, M. J. A. and Linoff, G.: „Data Mining Techniques for Marketing, Sales, and Customer Support", New York: Wiley, 1997

[BM98] L. Douglas Baker, Andrew McCallum: „Distributional Clustering of Words for Text Classification", SIGIR 1998: 96-103

[BU02] Cristopher J.C. Burges: „A Tutorial on Support Vector Machines for Pattern Recognition", Kluwer Academic Publishers, Boston 2002

[CA97] Peter Cabena et al.: „Discovering Data Mining: From Concept to Implementation", Prentice Hall, 1997

[DKM02] S. Inderjit Dhillon, Rahul Kumar, Subramanyam Mallela: „Enhancded word clustering for hierarchical text classification", KDD 2002: 191–200.

[EKSX96] ESTER, Martin; KRIEGEL, Hans-Peter; SANDER, Jörg; XU, Xiaowei: „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", München, 1996

[ES00] Martin Ester, Jörg Sander: "Knowledge Discovery in Databases – Techniken und Anwendungen", Springer Verlag, München 2000

[FE03] Reginald Ferber: „Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web“, dpunkt.verlag, Heidelberg 2003

[FPS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: „From data mining to knowledge discovery: An overview”, AAAI/MIT Press, 1996

[FPSU] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy: „Advances in Knowledge Discovery and Data Mining”, AAAI/MIT Press

[FR02] A. A. Freitas: „Data Mining and Knowledge Discovery with Evolutionary Algorithms”, Springer Verlag, Berlin 2002

[GG94] H. Genther, Manfred Glesner: „Automatic generation of a fuzzy classification system using fuzzy clustering methods”, SAC 1994: 180-183

[GM04] Mathias Goller: „Entwurf der Dissertation“, Private Kommunikation, Institut für Wirtschaftsinformatik, Abteilung Data & Knowledge Engineering, 2004

[HKKM97] Eui-Hong Han, George Karypis, Vipin Kumar, Bamshad Mobasher: „Clustering Based On Association Rule Hypergraphs”, DMKD 1997

[HK01] Jaiwei Han, Micheline Kamber: „Data Mining – Concepts and Techniques“, Morgan Kaufmann Publishers, San Francisco 2001

[KHK99] G. Karypis, E.-H. Han, and V. Kumar: „CHAMELEON: A hierarchical clustering algorithm using dynamic modeling”, IEEE Computer, 32(8):68-75, 1999.

[LSW97] Brian Lent; N. Arun Swami, Jennifer Widom: „Clustering Association Rules", ICDE 1997: 220–231

[LXY00] Bing Liu, Yiyuan Xia, Philip S. Yu: „Clustering Through Decision Tree Construction", CIKM 2000, VA USA 2000

[LY00] Hsiangchu Lai, Tzyy-Ching Yang: „A Group-based Inference Approach to Customized Marketing on the Web – Integrating Clustering and Association Rules Techniques", HICSS 2000

[MAR96] Manish Mehta, Rakesh Agrawal, Jorma Rissanen: „SLIQ: A Fast Scalable Classifier for Data Mining", IBM Almaden Research Center 650 Harry Road, San Jose, CA 95120, 1996

[MQ67] J. MacQueen: „Some Methods for Classification and Analysis of Multivariate Observations", 5th Berkeley Symp. Math. Statist. Prob., Volume 1, S 281-297, 1967

[RGG98] Raghu Ramakrishnan, J. Gehrke, V. Ganti: „RainForest – A Framework for Fast Decision Tree Construction of Large Datasets", VLDB-98, 1998

[SD01] Kai-Uwe Sattler, Oliver Dunemann: „SQL Database Primitives for Decision Tree Classifiers", CIKM-01 Atalanta, GM USA 2001

[SK04] Klaus Stöttinger: „Kombiniertes Data Mining - Effiziente Generierung von Hilfsinformationen während des Clustering", Institut für Wirtschaftsinformatik, Abteilung Data & Knowledge Engineering, 2004

[SM00] Padhraic Smyth: „Data Mining: Data Analysis on a Grand Scale?", Information and Computer Science University of California, Irvine 2000

[VHG03] Michalis Vazirgiannis, Maria Halkidi, Dimitrios Gunopulos: „Uncertainty Handling and Quality Assessment in Data Mining", Springer Verlag, 2003

[ZRL96] Tian Zhang, Raghu Ramakrishnan, Miron Livny: „BIRCH: An Efficient Data Clustering Method for Very Large Databases", SIGMOD '96 6/96 Montreal, Canada

Anhang

Testreihe 1: 5 numerische Dimensionen, 5 kategorische Dimensionen und zufällige Punkte als Trainingsdaten

55_db_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	23								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	71	134	208	350	702	1410	2127	3538	711
falsch klassifizierte Tupel	29	66	92	150	298	590	873	1462	289
Präzision	0,31	0,36	0,35	0,41	0,42	0,42	0,43	0,45	-1
Vollständigkeit	0,76	0,80	0,82	0,83	0,83	0,83	0,83	0,84	-1
Genauigkeit	0,44	0,51	0,5	0,52	0,53	0,53	0,54	0,55	-1
Klassifikationsgenauigkeit	0,71	0,67	0,69	0,70	0,70	0,71	0,71	0,71	0,71
Tatsächlicher Klassifikationsfehler	0,29	0,33	0,31	0,30	0,30	0,29	0,29	0,29	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,29

55_db_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	21								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	67	128	191	320	651	1298	1960	3269	654
falsch klassifizierte Tupel	33	72	109	180	349	702	1040	1731	346
Präzision	0,33	0,38	0,38	0,44	0,45	0,46	0,47	0,49	-1
Vollständigkeit	0,76	0,80	0,82	0,83	0,83	0,83	0,83	0,84	-1
Genauigkeit	0,48	0,54	0,55	0,58	0,58	0,59	0,59	0,60	-1
Klassifikationsgenauigkeit	0,67	0,64	0,64	0,64	0,65	0,65	0,65	0,65	0,65
Tatsächlicher Klassifikationsfehler	0,33	0,36	0,36	0,36	0,35	0,35	0,35	0,35	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,35

55_db_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	18								
Höhe des Baumes	4								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	73	137	214	359	715	1439	2167	3596	718
falsch klassifizierte Tupel	27	63	86	141	285	561	833	1404	282
Präzision	0,29	0,33	0,32	0,38	0,39	0,39	0,40	0,42	-1
Vollständigkeit	0,72	0,75	0,78	0,78	0,78	0,79	0,79	0,79	-1
Genauigkeit	0,40	0,47	0,45	0,48	0,49	0,49	0,49	0,50	-1
Klassifikationsgenauigkeit	0,73	0,69	0,71	0,72	0,72	0,72	0,72	0,72	0,72
Tatsächlicher Klassifikationsfehler	0,27	0,31	0,29	0,28	0,28	0,28	0,28	0,28	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,28

Testreihe 2: 5 numerische Dimensionen, 5 kategorische Dimensionen und nahe Punkte als Trainingsdaten

55_near_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	25								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1040
richtig klassifizierte Tupel	86	165	250	411	842	1657	2478	4115	947
falsch klassifizierte Tupel	14	35	50	89	158	343	522	885	93
Präzision	0,26	0,27	0,26	0,31	0,32	0,32	0,33	0,34	-1
Vollständigkeit	0,76	0,75	0,74	0,74	0,76	0,74	0,74	0,74	-1
Genauigkeit	0,29	0,33	0,31	0,35	0,35	0,35	0,35	0,36	-1
Klassifikationsgenauigkeit	0,86	0,83	0,83	0,84	0,84	0,83	0,83	0,82	0,91
Tatsächlicher Klassifikationsfehler	0,14	0,17	0,17	0,16	0,16	0,17	0,17	0,18	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,09

55_near_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	41								
Höhe des Baumes	6								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	81	155	232	379	782	1528	2307	3840	820
falsch klassifizierte Tupel	19	45	68	121	218	472	693	1160	220
Präzision	0,27	0,29	0,28	0,34	0,35	0,35	0,36	0,37	-1
Vollständigkeit	0,76	0,75	0,74	0,74	0,76	0,74	0,75	0,75	-1
Genauigkeit	0,34	0,36	0,37	0,41	0,40	0,41	0,41	0,42	-1
Klassifikationsgenauigkeit	0,81	0,78	0,77	0,76	0,78	0,76	0,77	0,77	0,79
Tatsächlicher Klassifikationsfehler	0,19	0,22	0,23	0,24	0,22	0,24	0,23	0,23	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,21

55_near_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	25								
Höhe des Baumes	4								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1040
richtig klassifizierte Tupel	83	157	238	393	809	1580	2356	3898	912
falsch klassifizierte Tupel	17	43	62	107	191	420	644	1102	128
Präzision	0,24	0,26	0,25	0,30	0,32	0,32	0,32	0,33	-1
Vollständigkeit	0,69	0,68	0,67	0,69	0,72	0,70	0,69	0,68	-1
Genauigkeit	0,28	0,33	0,31	0,35	0,35	0,35	0,36	0,36	-1
Klassifikationsgenauigkeit	0,83	0,79	0,79	0,79	0,81	0,79	0,79	0,78	0,88
Tatsächlicher Klassifikationsfehler	0,17	0,21	0,21	0,21	0,19	0,21	0,21	0,22	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,12

Testreihe 3: 5 numerische Dimensionen, 5 kategorische Dimensionen und entfernte Punkte als Trainingsdaten

55_far_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	25								
Höhe des Baumes	4								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1040
richtig klassifizierte Tupel	71	134	199	325	656	1265	1885	3109	710
falsch klassifizierte Tupel	29	66	101	175	344	735	1115	1891	297
Präzision	0,18	0,16	0,15	0,18	0,18	0,18	0,18	0,19	-1
Vollständigkeit	0,44	0,35	0,33	0,34	0,34	0,31	0,31	0,32	-1
Genauigkeit	0,26	0,24	0,23	0,24	0,23	0,23	0,23	0,24	-1
Klassifikationsgenauigkeit	0,71	0,67	0,66	0,65	0,66	0,63	0,63	0,62	0,71
Tatsächlicher Klassifikationsfehler	0,29	0,33	0,34	0,35	0,34	0,37	0,37	0,38	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,29

55_far_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	36								
Höhe des Baumes	7								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1007
richtig klassifizierte Tupel	69	129	187	289	577	1130	1689	2756	629
falsch klassifizierte Tupel	31	71	113	211	423	870	1311	2244	378
Präzision	0,16	0,15	0,14	0,15	0,15	0,15	0,15	0,16	-1
Vollständigkeit	0,38	0,32	0,30	0,26	0,25	0,23	0,23	0,23	-1
Genauigkeit	0,24	0,25	0,26	0,25	0,25	0,24	0,24	0,25	-1
Klassifikationsgenauigkeit	0,69	0,65	0,62	0,58	0,58	0,57	0,56	0,55	0,62
Tatsächlicher Klassifikationsfehler	0,31	0,35	0,38	0,42	0,42	0,43	0,44	0,45	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,38

55_far_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	25								
Höhe des Baumes	4								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1040
richtig klassifizierte Tupel	71	134	199	326	658	1269	1890	3115	716
falsch klassifizierte Tupel	29	66	101	174	342	731	1110	1885	291
Präzision	0,18	0,16	0,15	0,18	0,19	0,18	0,19	0,19	-1
Vollständigkeit	0,45	0,35	0,33	0,34	0,34	0,32	0,32	0,32	-1
Genauigkeit	0,26	0,24	0,23	0,24	0,23	0,23	0,23	0,24	-1
Klassifikationsgenauigkeit	0,71	0,67	0,66	0,65	0,66	0,63	0,63	0,62	0,71
Tatsächlicher Klassifikationsfehler	0,29	0,33	0,34	0,35	0,34	0,37	0,37	0,38	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,29

Testreihe 4: 10 numerische Dimensionen und zufällige Punkte als Trainingsdaten

10_db_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	9								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	74	139	209	350	721	1420	2154	3561	723
falsch klassifizierte Tupel	26	61	91	150	279	580	846	1439	277
Präzision (precision)	0,32	0,36	0,32	0,33	0,34	0,33	0,34	0,35	-1
Vollständigkeit (recall)	1,00	0,98	0,99	0,98	0,99	0,99	0,99	0,99	-1
Genauigkeit (accuracy)	0,50	0,55	0,52	0,53	0,53	0,53	0,52	0,53	-1
Klassifikationsgenauigkeit	0,74	0,70	0,70	0,70	0,72	0,71	0,72	0,71	0,72
Tatsächlicher Klassifikationsfehler	0,26	0,30	0,30	0,30	0,28	0,29	0,28	0,29	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,28

10_db_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	13								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	76	152	232	401	811	1576	2368	3939	787
falsch klassifizierte Tupel	24	48	68	99	189	424	632	1061	213
Präzision	0,32	0,34	0,29	0,30	0,31	0,30	0,31	0,32	-1
Vollständigkeit	1,00	1,00	1,00	1,00	1,00	0,99	0,99	0,99	-1
Genauigkeit	0,48	0,50	0,45	0,44	0,44	0,45	0,45	0,46	-1
Klassifikationsgenauigkeit	0,76	0,76	0,77	0,80	0,81	0,79	0,79	0,79	0,79
Tatsächlicher Klassifikationsfehler	0,24	0,24	0,23	0,20	0,19	0,21	0,21	0,21	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,21

10_db_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	5								
Höhe des Baumes	3								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1000
richtig klassifizierte Tupel	62	122	184	316	646	1261	1904	3168	646
falsch klassifizierte Tupel	38	78	116	184	354	739	1096	1832	354
Präzision	0,39	0,42	0,37	0,38	0,39	0,38	0,39	0,39	-1
Vollständigkeit	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,99	-1
Genauigkeit	0,62	0,65	0,61	0,61	0,61	0,61	0,61	0,61	-1
Klassifikationsgenauigkeit	0,62	0,61	0,61	0,63	0,65	0,63	0,63	0,63	0,65
Tatsächlicher Klassifikationsfehler	0,38	0,39	0,39	0,37	0,35	0,37	0,37	0,37	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,35

Testreihe 5: 10 numerische Dimensionen und nahe Punkte als Trainingsdaten

10_near_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	11								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	961
richtig klassifizierte Tupel	83	164	253	427	852	1689	2546	4196	781
falsch klassifizierte Tupel	17	36	47	73	148	311	454	804	180
Präzision (precision)	0,27	0,27	0,25	0,26	0,27	0,26	0,27	0,28	-1
Vollständigkeit (recall)	0,92	0,92	0,93	0,93	0,93	0,93	0,94	0,93	-1
Genauigkeit (accuracy)	0,37	0,40	0,35	0,35	0,36	0,36	0,37	0,38	-1
Klassifikationsgenauigkeit	0,83	0,82	0,84	0,85	0,85	0,84	0,85	0,84	0,81
Tatsächlicher Klassifikationsfehler	0,17	0,18	0,16	0,15	0,15	0,15	0,15	0,16	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,19

10_near_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	25								
Höhe des Baumes	11								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	961
richtig klassifizierte Tupel	78	157	240	414	830	1626	2444	4063	765
falsch klassifizierte Tupel	22	43	60	86	170	374	556	937	196
Präzision	0,31	0,32	0,28	0,29	0,30	0,29	0,30	0,31	-1
Vollständigkeit	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,99	-1
Genauigkeit	0,46	0,47	0,43	0,41	0,42	0,42	0,43	0,43	-1
Klassifikationsgenauigkeit	0,78	0,79	0,80	0,83	0,83	0,81	0,81	0,81	0,80
Tatsächlicher Klassifikationsfehler	0,22	0,21	0,20	0,17	0,17	0,19	0,19	0,19	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,20

10_near_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	11								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	961
richtig klassifizierte Tupel	75	149	230	395	758	1521	2284	3791	733
falsch klassifizierte Tupel	25	51	70	105	242	479	716	1209	228
Präzision	0,20	0,22	0,18	0,20	0,20	0,19	0,19	0,20	-1
Vollständigkeit	0,63	0,65	0,60	0,66	0,59	0,59	0,59	0,59	-1
Genauigkeit	0,31	0,33	0,28	0,29	0,29	0,28	0,28	0,29	-1
Klassifikationsgenauigkeit	0,75	0,75	0,77	0,79	0,76	0,76	0,76	0,76	0,76
Tatsächlicher Klassifikationsfehler	0,25	0,25	0,23	0,21	0,24	0,24	0,24	0,24	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,24

Testreihe 6: 10 numerische Dimensionen und entfernte Punkte als Trainingsdaten

10_far_1 - Splitstrategie ohne Verteilungsinformation

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	12								
Höhe des Baumes	6								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1048
richtig klassifizierte Tupel	87	171	261	438	869	1719	2597	4298	768
falsch klassifizierte Tupel	13	29	39	62	131	281	403	702	280
Präzision (precision)	0,28	0,30	0,26	0,27	0,29	0,28	0,28	0,29	-1
Vollständigkeit (recall)	1,00	1,00	1,00	1,00	0,99	0,99	0,99	0,98	-1
Genauigkeit (accuracy)	0,37	0,40	0,36	0,36	0,38	0,37	0,37	0,38	-1
Klassifikationsgenauigkeit	0,87	0,86	0,87	0,88	0,87	0,86	0,87	0,86	0,73
Tatsächlicher Klassifikationsfehler	0,13	0,14	0,13	0,12	0,13	0,14	0,13	0,14	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,27

10_far_2 - Min/Max-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	27								
Höhe des Baumes	11								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1048
richtig klassifizierte Tupel	43	85	133	220	424	849	1270	2080	546
falsch klassifizierte Tupel	57	115	167	280	576	1151	1730	2920	502
Präzision	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-1
Vollständigkeit	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	-1
Genauigkeit	0,33	0,32	0,33	0,32	0,32	0,34	0,33	0,33	-1
Klassifikationsgenauigkeit	0,43	0,43	0,44	0,44	0,42	0,42	0,42	0,42	0,52
Tatsächlicher Klassifikationsfehler	0,57	0,57	0,56	0,56	0,58	0,58	0,58	0,58	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,48

10_far_3 - Dichtefunktion-Schnittpunkt-Strategie

Kennzahl	Anzahl der Testtupel								
	100	**200**	**300**	**500**	**1000**	**2000**	**3000**	**5000**	**Trainingsdaten**
Anzahl der Knoten	11								
Höhe des Baumes	5								
Anzahl der Testtupel	100	200	300	500	1000	2000	3000	5000	1048
richtig klassifizierte Tupel	69	132	208	350	687	1385	2081	3447	722
falsch klassifizierte Tupel	31	68	92	150	313	615	919	1553	326
Präzision	0,12	0,11	0,08	0,08	0,10	0,09	0,10	0,10	-1
Vollständigkeit	0,33	0,29	0,24	0,24	0,27	0,27	0,27	0,28	-1
Genauigkeit	0,23	0,24	0,19	0,19	0,20	0,20	0,19	0,20	-1
Klassifikationsgenauigkeit	0,69	0,66	0,69	0,70	0,69	0,69	0,69	0,69	0,69
Tatsächlicher Klassifikationsfehler	0,31	0,34	0,31	0,30	0,31	0,31	0,31	0,31	-1
Beobachteter Klassifikationsfehler	-1	-1	-1	-1	-1	-1	-1	-1	0,31