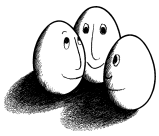
UNIVERSITÄT DORTMUND
Fachbereich Informatik

Lehrstuhl VIII
Künstliche Intelligenz

Projektgruppe 445
KDD Cup - Wettbewerb
Wissensentdeckung

Abschlussbericht

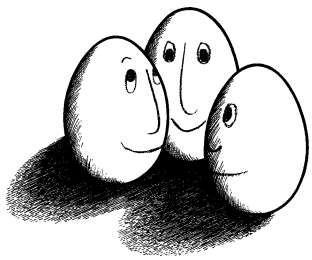


UNIVERSITÄT DORTMUND
Fachbereich Informatik

Lehrstuhl VIII
Künstliche Intelligenz

Projektgruppe 445
KDD Cup - Wettbewerb Wissensentdeckung

Abschlussbericht



Teilnehmer:
Dirk Dach
Holger Flick
Christophe Foussette
Marcel Gaspar
Daniel Hakenjos
Felix Jungermann
Christian Kullmann
Anna Litvina
Lars Michele
Siehyun Strobel
Marc Twiehaus
Nazif Veliu

Betreuer:
Prof. Dr. Katharina Morik
Dipl.-Inform. Martin Scholz

Danksagung

Die Teilnehmer der Projektgruppe 445 bedanken sich herzlich bei Katharina Morik und Martin Scholz für die Betreuung im Wintersemester 2003/04 und Sommersemester 2004. Ein ganz besonderer Dank gilt den Mitarbeitern des Lehrstuhls 8, die uns während der Durchführung der Projektgruppe zur Seite standen. Hervorzuheben ist Ingo Mierswa für seinen hervorragenden YALE-Support.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

Wissensentdeckung ist das Auffinden von interessanten und potenziell nützlichen Mustern in (sehr) großen Datenbeständen [25]. Das Gebiet hat seine Wurzeln im Bereich des maschinellen Lernens, der Datenbanken und in der Statistik. Der Prozess, der die gewünschten Ergebnisse liefert, besteht aus Dateninspektion, Datenbereinigung, Merkmalsextraktion, Merkmalsgenerierung, Merkmalsauswahl, Datentransformationen und Datenauswahl, bevor die Daten für ein ausgewähltes Lernverfahren (data mining) geeignet sind. Die Analyseergebnisse werden nach verschiedenen Kriterien (z.B. recall und precision) bewertet. Wissensentdeckung wird sehr vielseitig eingesetzt, beispielsweise im customer relationship management, für das Filtern von spam mails, für die Analyse von Gen-Daten oder auch für Verkaufsprognosen. Die Nachfrage nach in diesem Bereich ausgebildeten Menschen ist besonders hoch.
Seit einigen Jahren gibt es einen internationalen Wettbewerb, den KDD-Cup, bei dem ein Datensatz und eine Analyseaufgabe publiziert werden. Aus den eingegangenen Lösungen werden die drei besten ausgewählt und bei der internationalen KDD-Tagung vorgestellt.

1.1 Der Wissensentdeckungsprozess

Die allgemeinen Schritte eines Wissensentdeckungsprozesses sind im cross industrial standard dargelegt [13], aber beispielsweise auch in [5].
Bei der Wissensentdeckung spielen verschiedene Aufgabenstellungen eine Rolle. Werden aus Daten Modelle induziert, um damit Aussagen über noch unbekannte Fälle ableiten zu können, so spricht man bei einer Zuordnung zu Klassen von einem Klassifikationsproblem, bei der Vorhersage quantitativer Eigenschaften von einem Regressionsproblem. Ein Beispiel für eine Klassifikationsaufgabe ist die Erkennung von spam mails, ein Vorhersagemodell für Verkaufszahlen zu erstellen, stellt eine Regressionsaufgabe dar.
Neben Vorhersagemodellen ist man hier auch an einer Beschreibung der gege-

benen Daten interessiert, um deren Eigenschaften besser verstehen zu können. Für Marketingzwecke ist etwa eine Segmentierung der Kunden in homogene Gruppen interessant. Das Finden solch homogenener Gruppen fällt unter den Begriff des Clustering. Bei der Subgruppenentdeckung geht es darum, in den Daten Gruppen (z.B. von Kunden oder Patienten) zu identifizieren, die sich gegenüber der Gesamtheit statistisch auffällig verhalten.
Schließlich stellt die Analyse von Zeitreihen und Sequenzen noch wichtige eigenständige Aufgaben dar.

1.2 Verfahren zur Wissensentdeckung

Für jede der oben genannten Analyseaufgaben gibt es eine Fülle bekannter Data Mining Verfahren. Bekannte Verfahren zur Klassifikation und Regression sind die Support Vector Machine [11] und Entscheidungsbaumlerner, z.B. C4.5 [59]. Ein einfaches, statistisches Verfahren zur Klassifikation ist Naive Bayes [52]. Eine Vielzahl auf Prädikatenlogik basierender Lernverfahren werden unter dem Begriff Induktive Logische Programmierung subsumiert. Häufig in einem Datensatz gemeinsam auftretende Attributwerte (z.B. Warenkorbanalyse) können mit Hilfe von Verfahren wie APRIORI [4] gefunden werden. Varianten dieses Verfahrens dienen auch dem Erkennen häufig auftretender sequentieller Muster . Eine gute Übersicht über klassische Lernverfahren bietet [52].
Vorverarbeitung der Daten ist eine wesentliche Voraussetzung des erfolgreichen Einsatzes von Data Mining Algorithmen. Unter Merkmalsextraktion und -generierung fällt die Gewinnung von zunächst nur implizit in den Daten enthaltenen Informationen. Merkmalsauswahl bezeichnet die manuelle oder automatische Entfernung von für den Data Mining Schritt irrelevanten Merkmalen. [50] gibt eine Übersicht über bekannte Verfahren.

1.3 Werkzeuge für die Wissensentdeckung

Es sind inzwischen einige Werkzeuge zur Unterstützung der Wissensentdeckung verfügbar. Für den data mining Schritt gibt es die Sammlung von in JAVA implementierten Verfahren weka [69]. Für die Vorverarbeitung gibt es das System MiningMart, das unterschiedliche Verfahren integriert und direkt auf Datenbanken (z.B. Oracle) zugreifen kann [53].

Teil I

Grundlagen

Kapitel 2

Der Wissens-entdeckungungsprozess in Datenbanken

Daniel Hakenjos

Das Gebiet der Wissensentdeckung in Datenbanken (KDD) wird immer populärer und ist in letzter Zeit sehr stark gewachsen. Große Datenmengen, die vielleicht nützliche Informationen beinhalten, werden untersucht. In der Regel sind diese Informationen nicht offensichtlich, sondern bedürfen einer nicht trivialen Suche. Traditionelle Methoden der Datenanalyse sind bei großen Datenmengen ineffizient. Neuere Methoden, insbesondere des Data Minings, kommen zum Einsatz und werden in der Wissensentdeckung unterschiedlich eingesetzt. Dabei gab es bisher keine gemeinsame Basis, auf der ein solcher Wissensentdeckungsprozess durchgeführt werden konnte. Alle Datenanalysten setzten mehr oder weniger ihre eigenen Vorstellungen des Prozesses um. In diesem Abschnitt soll ausgehend von einer Definition ein Prozessmodell vorgestellt werden.

2.1 Was ist KDD?

In der Literatur lassen sich unterschiedliche Definitionen der Wissensentdeckung finden. Ihnen allen ist gemein, dass es aufgrund der riesigen Datenmengen eine nicht triviale Angelegenheit ist. Die folgende Definition von Fayyad wird die Einführung des Prozessmodells erleichtern.

Defintion: *"KDD is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."*[1]

[1] siehe [26] Seite 40

"process" impliziert, dass es sich um viele Schritte handelt, die nacheinander durchlaufen werden können bzw. wiederholt werden können. Dabei ist noch völlig offen gelassen in welcher Art und Weise diese Schritte nacheinander ausgeführt werden. Mit *"pattern"* sind Muster, Modelle und Strukturen in den Daten gemeint. Diese sind in einer geeigneten Sprache auszudrücken und beschreiben eine Teilmenge der Daten bzw. entdeckte Eigenschaften der Daten. Dieser Prozess ist nicht trivial (*"nontrivial"*), da er die Suche nach den Mustern, Modellen, Strukturen und Parametern mit einbezieht. Die entdeckten Muster müssen auch mit einer gewissen Wahrscheinlichkeit(*"valid"*) auf die Daten zutreffen. Es ist dem Datenanalysten überlassen in welcher Hinsicht er die Gültigkeit festlegt. Das entdeckte Muster muss ein Neuartiges(*"novel"*) sein. Vorzugsweise sollte es neuartig für den Benutzer sein, der dieses interpretiert. Zumindest aber neuartig für das System. Als Beispiel sei ein Neuronales Netz aufgeführt, das mit Hilfe einer Hyperebene die Klassifikation vornimmt. Es ist klar, dass mehrere solcher Hyperebenen existieren können, wobei jede neue trennende Hyperebene für das System als neuartig gilt, aber für den Benutzer nur bedingt als neuartig angesehen werden kann. Insbesondere ist man daran interessiert solche Erkenntnisse über die Daten zu gewinnen, die einen potentiellen Nutzen darstellen (*"potentially useful"*). Dabei sollen die Muster auch verständlich(*"understandable"*) sein. Dazu ist oft eine Nachbearbeitung notwendig.[2]

In der Literatur werden die Begriffe Data Mining und KDD oft synonym gebraucht, was zu Missverständnissen führt. Data Mining ist nur ein Teilprozess des Wissensentdeckungsprozesses. Dies wird auch in den nachfolgenden Abschnitten deutlich.

2.2 Ein Prozessmodell

2.2.1 CRISP-DM 1.0

CRISP-DM (**CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining) wurde 1996 von drei erfahrenen Unternehmen des jungen und stark wachsenden Data-Mining-Marktes gegründet. Dies waren DaimlerChrysler, SPSS und NCR. Aus den schon in der Einführung dieses Abschnittes genannten Gründen hatten sie sich zusammengeschlossen und ein Jahr später ein Konsortium gegründet. Dieses hatte die Zielsetzung CRISP-DM industrie-, werkzeug- und applikationsneutral zu gestalten. Die Special Interest Group (SIG) wurde ins Leben gerufen. Resultat des ersten Workshops war, dass die Notwendigkeit eines solchen Prozessmodells für die Industrie von besonderer Wichtigkeit ist. Obwohl es Unterschiede in der Terminologie und der Abgrenzung der Phasen gab, war

[2] vergl. [6]

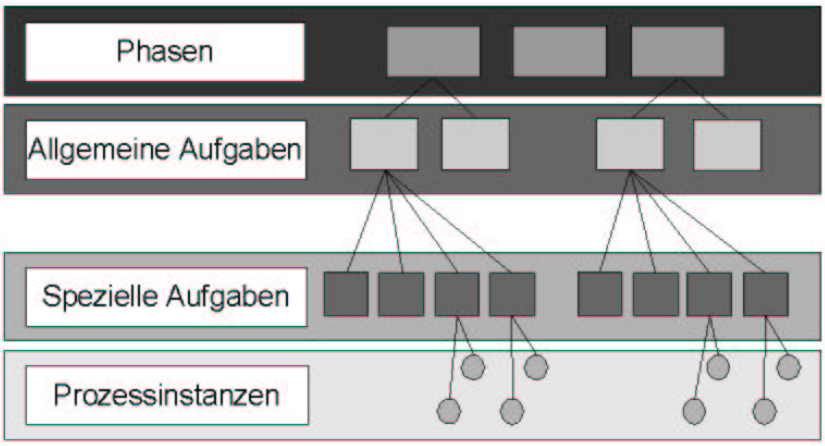


Abbildung 2.1: hierarchische Methodik von CRISP-DM 1.0 – siehe [13] Seite 9

eine gemeinschaftliche Basis in der Betrachtung der Phasen vorhanden. Innerhalb der nächsten $2\frac{1}{2}$ Jahre wurde das Modell entwickelt. Dazu trugen auch zahlreiche Workshops und Tagungen bei, so dass Mitte 1999 ein erster Entwurf vorlag. Das Prozessmodell wurde in der darauf folgenden Zeit erfolgreich auf verschiedene Applikationen übertragen und lag mit kleinen Änderungen im August 2000 vor. In dieser Form wird es auch im nächsten Abschnitt dargestellt.

Das Prozessmodell zeichnet sich dadurch aus, dass es nicht in theoretischer oder akademischer Manier entwickelt wurde. Der Erfolg ist darin begründet, dass es auf den Erfahrungen der Vergangenheit insbesondere der Durchführung von Data-Mining-Projekten basiert. Aufgrund dieser Allgemeingültigkeit und Akzeptanz, die dieses Modell findet, soll es hier vorgestellt werden.

2.2.2 Hierarchische Methodik

Hinter dem Modell steht eine hierarchische Methodik, wie sie in der obigen Abbildung zu erkennen ist. Zu den einzelnen Phasen existieren allgemeine Aufgaben, die weiter spezialisiert sind. Erst hieraus ergeben sich die Prozessinstanzen für die aktuelle Aufgabenstellung. In der zweiten Ebene werden die Aufgaben so allgemein wie möglich beschrieben, um alle möglichen Situationen abzudecken. In der Praxis können viele der Aufgaben in unterschiedlicher Reihenfolge vorkommen. Die Rückkehr zu vorherigen Aufgaben und die Wiederholung dieser gehört ebenso zur Beschreibung.

2.3 Phasen des Modells

2.3.1 Überblick

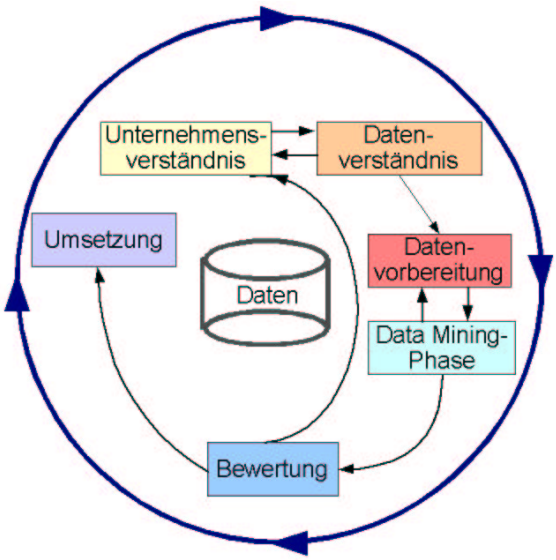


Abbildung 2.2: Zusammenspiel der Phasen – siehe [13] Seite 13

Der Standardprozess ist, wie in der Abbildung zu sehen, in sechs Phasen aufgeteilt: Unternehmensverständnis, Datenverständnis, Datenvorbereitung, Data-Mining-Phase, Bewertung und Umsetzung.

Das Unternehmensverständnis ist die Startphase des Projektes und bezieht sich auf die Ziele und Anforderungen der Wissensentdeckung. Ausgehend von der Perspektive des Anwenders wird das Problem definiert, ein vorbereitender Plan erstellt und dieses bisherige Wissen in eine Data-Mining-Problemstellung überführt.

In der zweiten Phase müssen Daten gesammelt werden. Dies schliesst ein, dass man sich mit den Daten vertraut macht. Dabei können erste Erkundungen vorgenommen werden. Ebenso kann die Qualität der Daten festgestellt werden.

Die Datenvorbereitung ist eine Phase, die im Wissensentdeckungsprozess an Bedeutung gewonnen hat. Die Daten müssen für die nachfolgende Data-Mining-Phase vorbereitet werden. Es hat sich herausgestellt, dass es von großer Bedeutung ist die Daten für die gewählte Data-Mining-Technik zu transformieren.

Die Data-Mining-Phase stellt die Kernphase des Prozesses dar. Es kommen die gewählten Techniken zum Einsatz, wobei es sich dabei oft um Techniken aus dem Maschinellen Lernen handelt.

Zur Prüfung der Gültigkeit des gewonnenen Modells dient die Bewertungsphase, insbesondere kommen dabei statistische Methoden zum Einsatz.

Zuletzt erfolgt die Umsetzung des Modells, dabei kann es sich schlicht um die einmalige Anwendung handeln oder aber auch um die Integrierung in ein System, das zukünftig gewartet und überwacht werden muss.

In der Abbildung sind die wesentlichen Verbindungen zwischen den Phasen gekennzeichnet. Die Phasen sind nicht starr angeordent, sondern das Hin- und Herspringen zwischen den Phasen ist sogar während des Prozesses erforderlich. Die Entscheidung dafür basiert auf den Resultaten der einzelnen Phasen. Wobei die Daten, aus denen das Wissen gewonnen werden soll im Mittelpunkt der ganzen Betrachtung stehen. Besonders die inneren vier Phasen sind stark an die Daten gekoppelt. Der äußere Kreis macht die zyklische und iterative Eigenschaft des Prozesses deutlich. Der Wissensentdeckungsprozess verläuft nicht "straight forward", sondern es werden immer wieder neue Erkenntnisse gewonnen, Einstellungen in der angewandten Data-Mining-Technik notwendig werden, so dass sich dieser Prozess dynamisch auffassen lässt.

2.3.2 Phase I – Unternehmensverständnis

Unternehmensziele bestimmen

Zunächst muss der Datenanalyst verstehen, was der Kunde erreichen möchte. Dazu ist es notwendig, dass er Informationen über das Unternehmen sammelt. Es müssen die primären Ziele des Kunden beschrieben und festgehalten werden. Ebenso muss ein Erfolgskriterium verhandelt werden. Der Analyst nimmt zunächst nur die Unternehmensperspektive ein. Es ist für die nachfolgenden Phasen von besonderer Wichtigkeit, dass ein großes Hintergrundwissen gesammelt wird. Eine Möglichkeit besteht durch Interview der Leiter der einzelnen bzw. betroffenen Unternehmensbereiche.

Zugriffssituation

Der nächste Schritt umfasst eine detaillierte Faktensuche über die vorhandenen Ressourcen, Bedingungen und Annahmen. Dazu wird das Ressourceninventar aufgelistet, welches personelle und maschinelle Dimensionen wie Hard- und Software berücksichtigt. In den Anforderungen des Projektes sind insbesondere ein Zeitplan und die Qualität des Resultates aufzunehmen. Ebenso kann

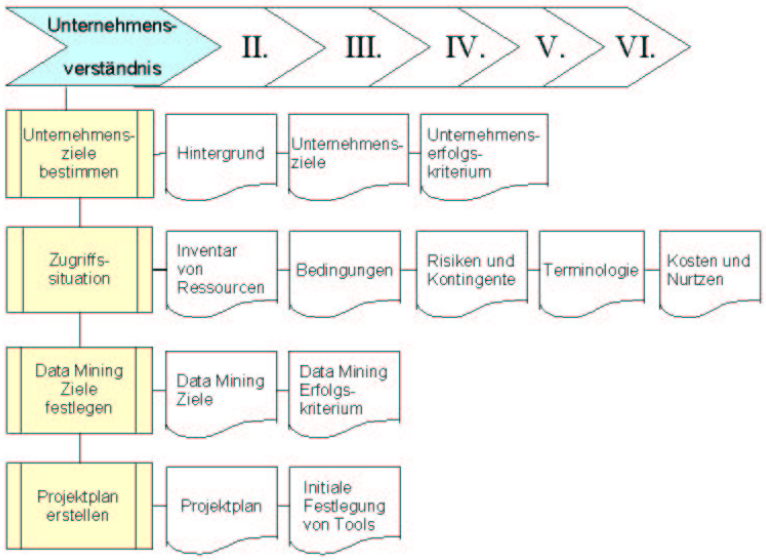


Abbildung 2.3: Phase I – Unternehmensverständnis – siehe [13] Seite 16

erweiternd zum Hintergrundwissen eine Terminologie aufgeführt werden. Diese kann Ausdrücke und Fremdwörter beinhalten, die aufgaben- und branchenspezifisch sind. Eine Kosten-Nutzen-Gegenüberstellung fördert die Einhaltung des gesetzten Projektrahmens.

Data Mining Ziele festlegen

Nun ist es an der Zeit die Data Mining Ziele zu formulieren, dabei nimmt der Analyst die Data Mining Perspektive ein. Das entsprechende Erfolgskriterium muss in technischer Sprache beschrieben werden. Gegenüber der akademischen Sicht ist die Ausprägung des Erfolgskriteriums problemabhängig und wird durch das Erfolgskriterium des Kunden festgelegt!

Projektplan erstellen

Hier werden die einzelnen Schritte des Projektes detailliert für jede Phase aufgeführt. Ebenso werden die analysierten Abhängigkeiten zwischen Zeitmanagement und Risiko aufgeführt. Es findet schon eine initiale Festlegung der zu verwendenden Werkzeuge und Techniken statt. Dies ist notwendig bevor spätere Änderungen das ganze Projekt beeinflussen.

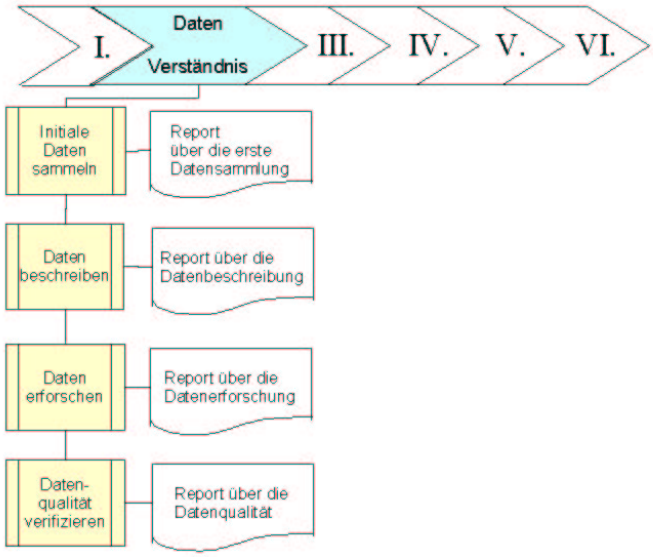


Abbildung 2.4: Phase II – Datenverständnis – siehe [13] Seite 20

2.3.3 Phase II – Datenverständnis

Initiale Daten sammeln

Zu Beginn dieser Phase müssen Daten gesammelt werden. In der jeweiligen Unternehmung gibt es verschiedene Ressourcen, aus denen die Daten zusammengestellt bzw. geladen werden. Dazu zählt auch, dass die Methoden der Akquirierung und dabei aufgetretene Probleme und zugehörige Lösungen dokumentiert werden.

Daten beschreiben

Nun werden die zugehörigen Metadaten generiert. Nach der Untersuchung der oberflächlichen Dateneigenschaften wird ein Report erstellt, der Informationen wie Datenformat, Datenqualität, Anzahl der Datensätze und Identifikation der Felder enthält. In den bisherigen KDD-Cups wurden diese Informationen teilweise zur Verfügung gestellt, sofern dies möglich war. Die nachfolgenden Schritte stellen die ersten Schritte im KDD-Cup dar.

Daten erforschen

Es beginnt mit der Erforschung der Daten hinsichtlich des Data Mining Zieles. Dafür kann/können die Verteilung der Schlüsselattribute, die Beziehungen zwischen den Attributen, einfache Aggregationen oder statistische Analysen durchgeführt werden. Der zugehörige Report beinhaltet erste Hypothesen und

deren Auswirkung auf das Projekt, Visualisierungen, interessante Datensätze für spätere Untersuchungen. Für die Datenerforschung hinsichtlich Verteilung und Visualisierung können auch bekannte Werkzeuge eingesetzt werden.

Datenqualität verifizieren

In der Praxis findet man die Daten oft in ungenügendem Zustand vor. Dieser Zustand muss erst verifiziert werden. Dies betrifft die Vollständigkeit der Daten, die Verrauschtheit der Daten und die Fehler in den Werten. Der Report listet diese Probleme auf und gibt Lösungsvorschläge. Hierbei ist von besonderer Bedeutung, dass in den ersten Schritten genügend Hintergrundwissen gesammelt wurde, denn die Lösungen für die Datenqualitätsprobleme sind vom Wissen über die Daten und das Unternehmen abhängig.

2.3.4 Phase III – Datenvorbereitung

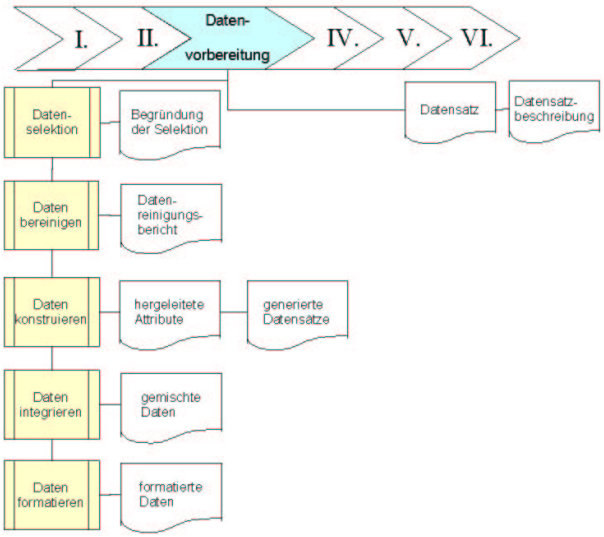


Abbildung 2.5: Phase III – Datenvorbereitung – siehe [13] Seite 23

Zwei Ausgaben

Diese Phase hat eine Besonderheit: Als einzige hat sie zwei Ausgaben. Diese sind der Datensatz und die dazugehörige Beschreibung, die durch die Datenvorbereitung erzeugt wird. Bezüglich der Datenbeschreibung sei auf die vorherige Phase verwiesen. Bezüglich der nachfolgenden allgemeinen Aufgaben sei im Detail auf den Abschnitt Merkmalsgenerierung und -selektion verwiesen. Dort

wird gezeigt wie mit Hilfe des Filter- bzw. Wrapperansatztes Datenselektion in Ausprägung der Attributauswahl durchgeführt werden kann.

Datenselektion

Entscheidungen über Gebrauch der Daten für die Analyse müssen getroffen werden. Kriterien dafür sind zum Beispiel das Data Mining Ziel, Qualitäts- und technische Bedingungen. Für eine detaillierte Dokumentation sind Gründe für die Einbeziehung bzw. den Ausschluss der Datensätze aufzuführen. Zu beachten ist, dass die Auswahl von Attributen ebenso wichtig wie die Auswahl der Datensätze ist. Es sei noch einmal auf den Filter- bzw. Wrapperansatz verwiesen. In der Datenbankwelt unterscheidet man eigentlich Selektion (Auswahl von Tupeln) und Projektion (Auswahl von Spalten). Im hier beschriebenen Schritt wird diese Unterscheidung nicht vorgenommen. Datenselektion ("data selection") umfasst hier beide Begriffe.

Daten bereinigen

Unter Nutzung der Ergebnisse der Verifizierung der Datenqualität werden reine Subdaten ausgewählt und fehlende Werte geschätzt. Der zugehörige Datenbereinigungsbericht beinhaltet Entscheidungen und Aktionen um die Qualitätsmängel zu beheben. Dazu kann es notwendig sein, dass Daten transformiert werden müssen.

Daten konstruieren

Es gibt viele Möglichkeiten Daten zu konstruieren. Die Herleitung kann über einfache Aggregationen erfolgen. Es ist auch die Produktion ganz neuer Datensätze oder die Transformation existierender Attribute möglich. Wie gewohnt muss dieses dokumentiert werden.

Daten integrieren

Die Datenintegration umfasst Methoden um Relationen miteinander zu kombinieren. Zum Beispiel werden Relationen, die Informationen über dieselben Objekte enthalten über den Verbund zusammen geführt. Zu diesen zusammengeführten Daten gehören auch Aggregationen.

Daten formatieren

Insbesondere für die Vorbereitung für die nächste Phase ist die Transformation der Daten notwendig, dies umfasst syntaktische Modifikationen, die die Semantik der Daten nicht verändert. Einige Werkzeuge erfordern besondere Datenformate, die erstellt werden müssen. Möglicherweise muss diese Transformation

für jedes Werkzeug bzw. jede Data Mining Technik explizit durchgeführt werden. Beispiele für solche Transformationen sind der Stemming-Algorithmus für Texte und die Umordnung der Attribute, so dass das Zielattribut zuletzt aufgeführt ist.

2.3.5 Phase IV – Data-Mining-Phase

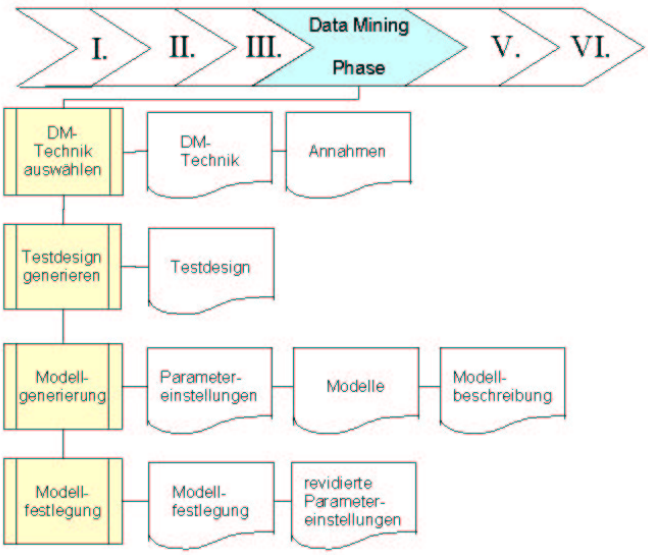


Abbildung 2.6: Phase IV – Data-Mining-Phase – siehe [13] Seite 27

Data Mining Technik auswählen

Phase IV ist die eigenliche Kernphase eines Data-Mining-Projektes. Es muss eine aktuelle Modellbildungstechnik ausgewählt werden, die in dieser Phase zum Einsatz kommen soll. In Phase 1.4 wurde schone eine initiale Festlegung der Werkzeuge getroffen. Diese gilt es hier genauer zu spezifizieren! Beispielsweise hat man sich für die Anwendung eines Entscheidungsbaumverfahrens entschlossen, dann muss man sich für ein konkretes Verfahren entscheiden: ID3 oder C4.5 (Entscheidungsbaumlernverfahren werden in Kapitel 4 vorgestellt). Es wird nicht nur die Auswahl des Werkzeuges dokumentiert, sondern auch die Voraussetzungen bezüglich der Daten, die dieses Werkzeug trifft. Dies ist insbesondere dann wichtig, wenn eine entsprechende Vorverarbeitung noch nicht stattgefunden hat.

Testdesign generieren

Danach wird ein sogenanntes Testsdesign generiert. Dies ist notwendig, um die Qualität und Gültigkeit des Modells zu bestimmen. In welchem Maß die Gültigkeit auf den Daten festzulegen ist, hängt von der Problemstellung ab. Es müssen also der beabsichtigte Plan zum Testen und Trainieren bestimmt werden. Dazu gehört auch festzulegen, wie die Daten in Trainings- und Testdaten geteilt werden. Eine Möglichkeit besteht in der Anwendung der n-facher Kreuzvalidierung, wobei in der Praxis oft die 10-fache Kreuzvalidierung zum Einsatz kommt. Dabei erfolgt die Datenteilung in 10 Teile, wobei immer ein Teil dem Training vorenthalten und zum Test eingesetzt wird.

Modellgenerierung

Nun kann die Generierung des Modells erfolgen. Dazu wird das gewählte Werkzeug auf den vorbereiteten Datensatz angewendet. In dieser allgemeinen Aufgabe ist es ganz wichtig die Parametereinstellungen und Gründe dafür zu dokumentieren. Die Modellerzeugung mittels des Werkzeuges oder einer Applikation kann natürlich wiederholt stattfinden. Es ist dabei zu beachten, dass alle Einstellungen und Änderungen dokumentiert werden. Dies ist besonders wichtig, da im nächsten Schritt eine Auswahl getroffen werden soll. Die gewonnenen Modelle werden abschließend beschrieben und interpretiert. Nicht immer ist eine Interpretation sofort möglich, insbesondere wenn das Modell sehr umfangreich ist. Probleme und offene Fragen werden ebenfalls dokumentiert.

Modellfestlegung

In der letzten allgemeinen Aufgabe dieser Phase muss der Datenanalyst sich auf ein Modell festlegen. Diese Festlegung erfolgt hinsichtlich des Data-Mining-Erfolgskriteriums und des Tests. Dabei kann ein Ranking der verschiedenen Modelle vorgenommen werden, dies kann auch geschehen, wenn unterschiedliche Werkzeuge zum Einsatz gekommen sind. Eventuell ist eine Wiederholung dieser Phase notwendig, da zum Beispiel das Erfolgskriterium noch nicht erreicht wurde. Die revidierten Parametereinstellungen bzw. -änderungen werden dokumentiert. Dieser Vorgang kann solange itereriert werden, bis ein bestmögliches Modell gefunden wurde.

2.3.6 Phase V – Bewertung

Resultate bewerten

Wurde ein bestmögliches Modell gefunden bzw. wurde ein Modell festgelegt, so muss dieses bewertet werden. Es muss festgelegt werden in welchem Maß

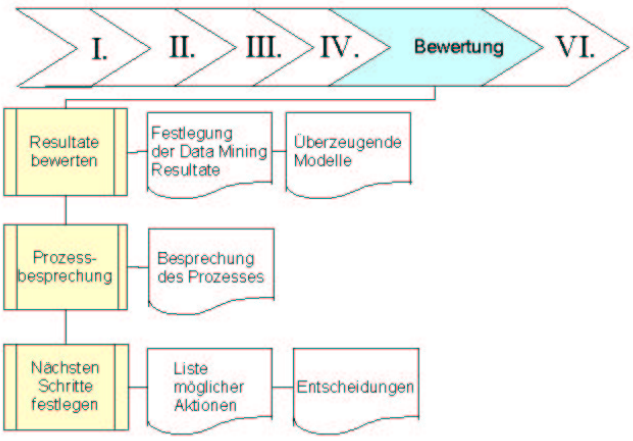


Abbildung 2.7: Phase V – Bewertung – siehe [13] Seite 30

es den Unternehmenszielen entspricht. Sind genügend Kapazitäten (zeitlich, personell, finanziell etc.) vorhanden, so kann ein Test in der vorgesehenen Applikation oder eine prototypische Umsetzung in der Praxis erfolgen. Die Resultate dieser Phase werden zusammengefasst und es wird Stellung bzgl. des Erreichens der Unternehmensziele bezogen. Modelle, die überzeugt haben, sprich das Erfolgskriterium erfüllen, werden in der Dokumentation aufgelistet.

Prozessbesprechung

Bevor die Entscheidung über das weitere Vorgehen getroffen werden kann, muss eine Besprechung des bisherigen Prozesses stattfinden. Es ist dabei zu überprüfen, ob nicht ein wichtiger Faktor vergessen oder übersehen wurde. Dazu können Fragestellungen wie "Wurde das Modell korrekt generiert?" oder "Sind die benutzten Attribute zugänglich, auch zukünftig?" dienen. Es wird abschließend zu dieser allgemeinen Aufgabe ein Rückblick erstellt, in dem fehlende Schritte und Wiederholungen hervorgehoben werden.

Nächsten Schritte festlegen

Nun kann eine Entscheidung über das weitere Vorgehen getroffen werden. Je nach Projektfortschritt hinsichtlich des Erfolgskriteriums kann ein Sprung in die vorherige Phase erfolgen, ein neues Projekt begonnen werden oder aber einfach in Phase VI fortgeführt werden. Alle potentiellen Aktionen werden mit dazugehöriger Einschätzung aufgelistet, um darauf aufbauend eine Entschei-

dung treffen zu können. Ganz besonders ist zu beachten, dass das Zeitkontingent die Entscheidung wesentlich beeinflussen kann.

2.3.7 Phase VI – Umsetzung

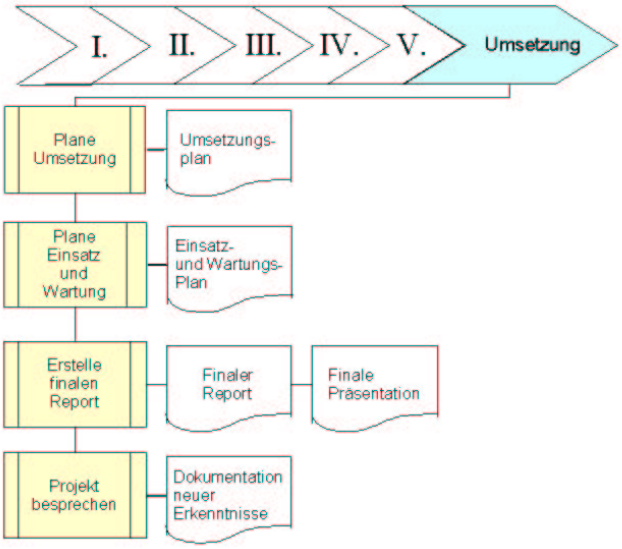


Abbildung 2.8: Phase VI – Umsetzung – siehe [13] Seite 32

Plane Umsetzung

In der letzten Phase erfolgt die Umsetzung in der Unternehmung. Bei größeren Projekten muss die Umsetzung geplant werden, dafür werden die einzelnen Schritte und wie diese erreicht werden sollen aufgeführt. Die Resultate der Bewertung dienen zum Aufstellen der Umsetzungstrategie.

Plane Einsatz und Wartung

Die Umsetzung, sprich der Einsatz der Data-Mining-Resultate in der Unternehmung, kann eine einmalige Angelegenheit sein. Im Gegensatz dazu kann aber auch ein langfristiger Einsatz möglich sein. Dies erfordert auch strategische Komponenten wie zum Beispiel Wartung und Überwachung. Man beachte aber hierbei, dass Überwachung und Wartung eigene Prozesse sein können.

Erstelle finalen Report

Aufbauend auf dem Umsetzungsplan wird der Projekt- und Erfahrungsbericht geschrieben. Dazu zählen eine zusammenhängende Präsentation der DM-

Resultate, vorherige Arbeitsergebnisse und Zusammenfassung der Resultate. Beim Kunden findet dann eine verbale Präsentation des Projektes statt.

Projekt besprechen

Zum Schluss erfolgt im Rahmen der Projektbeteiligten eine Besprechnung. In dieser soll herausgestellt werden was gut, was schlecht gelaufen ist und was verbessert werden kann. Wichtige Erfahrungen und Entdeckungen im Umgang mit den Werkzeugen und Daten während des Projektes werden dokumentiert, um für spätere Projekte dieses zur Verfügung zu stellen.

2.3.8 Kritische Betrachtung

Wie im Abschnitt über die Phasen des Modells erwähnt ist CRISP-DM 1.0 als dynamisch anzusehen. Jedoch entsteht beim Betrachten der allgemeinen und speziellen Aufgaben nicht immer dieser Eindruck. Beispielsweise wird der dynamische Zusammenhang zwischen erster und zweiter Phase nicht ganz deutlich! Wann wird ein Rückschritt in Phase I notwendig?
Der Zusammenhang zwischen der Datenvorbereitung und der Data-Mining-Phase ist auch nicht abschliessend geklärt. Aus dem Standardprozess entsteht der Eindruck, dass die Selektion der Attribute und Datensätze unabhängig vom der anzuwendenden Data-Mining-Technik geschieht. Jedoch ist es in der Praxis anders: Bei der Anwendung des Wrapper-Ansatzes zur Attributselektion kommt während der Selektion die ausgewählte Data-Mining-Technik schon zum Einsatz. Sprich die Attributauswahl geschieht zielgerichtet bezüglich der Data-Mining-Technik. Sicherlich kann es nicht Ziel eines Standardprozesses sein, dass alle möglichen Spezialfälle insbesondere wirklich alle zukünftigen Anwendungsmöglichkeiten erfasst werden. Dazu ist es notwendig das Prozessmodell in der Zukunft immer wieder den Bedürfnissen der Praxis anzupassen. Der Nachteil ist vielleicht gerade deshalb festzustellen, da es nicht theoretisch entwickelt wurde, sondern auf den Erfahrungen, wie bisher Data-Mining-Projekte durchgeführt wurden, beruht.

2.4 Eigenschaften des Prozessmodells

Anhand des nun aufgezeigten Prozessmodells können einige Eigenschaften bestimmt werden. Offensichtlich ist der Wissensentdeckungsprozess interdisziplinär. Es sind vielfältige Kenntnisse notwendig. Dazu zählen unter anderem Kenntnisse aus den Gebieten Maschinelles Lernen, Mustererkennung, Informationssysteme und Datenbanken, Statistik, Wissenserwerb für Expertensysteme und Datenvisualisierung. Der Wissensentdeckungsprozess ist mehrfach iterativ und deshalb gestaltet er sich oft langwierig. Dazu sind komplexe Interaktionen

notwendig und oft ein mehr oder minder großes Hintergrundwissen. [3].

Der Wissensentdeckungsprozess wird als "human-centered" beschrieben [4], dies ist eigentlich nicht selbstverständlich! Den ganzen Prozess maschinell durchzuführen ist nicht möglich, da der Datenanalyst den Prozess mit seinem Wissen über das Problem und die angewendeten Techniken steuert. Dazu gehört auch die Inspektion von Zwischenergebnissen, die Bewertung dieser, sowie immer wieder das Vorgehen nach der Trial-and-Error-Methode.

2.5 Ähnliche Modelle

In der Literatur findet man eine Vielzahl ähnlicher Modelle, die versuchen den Sachverhalt ähnlich darzustellen. Als Beispiel möchte ich hier das Prozessmodel nach Fayyad, wie es auch in der Abbildung 2.9 zu sehen ist, anführen. Aus beiden Beispielen wird deutlich, dass die Darstellung des Prozesses komplizierter und ebenso auch oberflächlicher und detaillierter werden kann. Vor allem ist eine umfassende Darstellung des Prozesses von der Zielsetzung des Unternehmens bis hin zur Umsetzung selten zu finden. Oft wird in der Beschreibung nur auf die Data-Mining-Phase fokussiert. In dieser Hinsicht ist CRISP-DM 1.0 ausgewogener und stellt die einzelnen Phasen in gleichverteilter Gewichtung dar. Sicherlich ist in der Praxis eine unterschiedliche Gewichtung der Phasen vorzufinden, dies hängt aber von der Problemstellung ab. Beim KDD-Cup wird sich die Projektdurchführung im wesentlichen auf Teile von Phase II, auf Phase III, Phase IV und Phase V beschränken, wobei die endgültige Bewertung des Modells projektextern durch ein Wettbewerbskomitee nach objektiven und/oder subjektiven Kriterien vorgenommen wird. Diese Fokussierung auf Datenselektion, Datenvorverarbeitung, Data Mining und Bewertung ist auch im Modell nach Fayyad (siehe Abbildung 2.9) zu sehen.

2.6 Überblick Lernaufgaben

Es gibt verschiedene Möglichkeiten Lernaufgaben zu charakterisieren und einzugrenzen. Zu nennen wäre die einfache binäre Unterscheidung zwischen überwachten und unüberwachten Lernaufgaben. Überwachte Lernverfahren beschäftigen mit dem Lernen von Hypothesen aus Beispielen, zu denen bereits Zielwert(e) gegeben sind. Jedoch lässt sich in der Literatur eine durchaus "feinere" Einteilung finden. Man unterscheidet im wesentlichen fünf Lernaufgaben: Klassifikation, Regression, Clustering, Assoziationsregeln und Subgruppenentdeckung.

[3] siehe [5] Seite 39 f
[4] siehe [5]

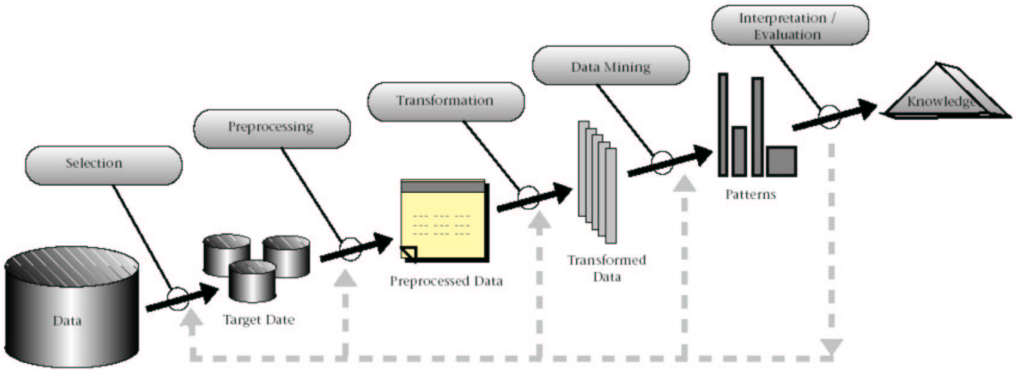


Abbildung 2.9: Überblick über Prozessschritte nach Fayyad – siehe [26] Seite 41

Lernaufgabe	**Lernverfahren**	**Spezifizierung in**
Klassifikation	SVM	Kapitel 5
Regression	Neuronale Netze BackPropagation	
Clustering	k-Means	Kapitel 10
Assoziationsregeln	Apriori	Kapitel 8 und 9
Subgruppenenntdeckung	MIDOS	Kapitel 7
Regellernen	RDT	Kapitel 6

Tabelle 2.1: Lernaufgabe und Lernverfahren

Um die einzelnen Lernaufgaben näher zu betrachten, wollen wir zunächst den Begriff der Lernaufgabe selbst spezifizieren. Ein Lernaufgabe wird definiert durch eine Beschreibung der zur Verfügung stehenden Eingaben, der erwarteten Ausgabe des Lernsystems und den Randbedingungen des Lernsystems selbst. Mit einer so definierten Lernaufgabe ist nun genau festgelegt, was es bedeutet, wenn ein Lernverfahren lernt.

Schauen wir uns zunächst das Lernen aus klassifizierten Beispielen – die Klassifikation – an. Bei dieser Lernaufgabe nimmt man an, dass es eine unbekannte Funktion gibt, die Objekte einer bestimmten Grundmenge einen Wert zuordent. Da es im menschlichen Zusammenleben viele Entscheidungsprobleme gibt, ist zunächst die Funktion als eine Trennende aufzufassen, sprich eine binäre Klassifikation zu lösen ist. Ebenso interessant und nicht nur von theoretischem Interesse ist das Gebiet der mehrwertigen Klassifikation. Bei der Klassifikation allgemein werden dem lernendem System Beispiele der gesuchten

Funktion vorgestellt. Jedes dieser Beispiele hat eine Beschreibung (Attribute) und einen Zielwert. Lernen bedeutet nun, dass das Verfahren eine Funktionsbeschreibung (Hypothese) ausgibt, mit der Beispiele mit noch unbekanntem Zielwert möglichst genau vorhergesagt werden.

In Analogie zur Klassifikation lässt sich die Regression definieren. Dabei ist aber der Funktion f nicht ein diskreter Bildbereich, sondern ein kontinuierlicher Bildbereich beispielsweise das Intervall $[0, 1]$ zugeordnet.

Klassifikation und Regression sind Klassen von Lernaufgaben, die zu der Klasse der überwachten Lernaufgaben gehören. Clustering ist gerade das Entgegengesetzte. Eine gegebene Instanzenmenge S aus einem Instenzenraum X, wird so in verschiedene Teilmengen bzw. Cluster $C_i \subseteq S$ $i = 1, ..., n$ aufgeteilt, so dass die Instanzen $s \in C_i$ möglichst ähnlich sind. Im Idealfall sollten sich die Cluster nicht überlappen:

$$\forall i \in [1, n] \forall j \in [1, n] C_i \cap C_j = \{\}$$

Im Gegensatz zur Subgruppenentdeckung ist zu bemerken, dass die Cluster nicht hinsichtlich ihrer Eigenschaft bezüglich ausgewählter Zielattribute identifiziert werden, sondern nur durch die Eigenschaft größtmöglicher Ähnlichkeit der Instanzen innerhallb eines Clusters.

Das Finden von Assoziationsregeln ist keine Spezialisierung des Funktionslernens aus Beispielen. Als Beispiel sei hier die sogenannte Warenkorbanalyse erwähnt. Mit Hilfe dieser ist es den Kaufhäusern und Discountern möglich interessante Zusammenhänge beim Kundenverhalten aufzudecken. Eine interessante Frage einer solchen Analyse kann zum Beispiel sein, welche Artikel unter welchen Bedingungen zusammen gekauft werden. Eine Regel könnte dann lauten: "Falls in einem Einkauf Zahnbürste(n) und Dusch-Gel gemeinsam gekauft werden, so ist es wahrscheinlich, dass auch Zahnpasta gekauft wird." Solche Assoziationsregeln können den einzelnen Entscheidern in der Unternehmung als Unterstützung dienen.

Bei der Subgruppenentdeckung ist es nicht von Interesse ein globales Modell zu lernen, sondern vielmehr lokale Beobachtungen vorzunehmen. Diese könnten zum Beispiel folgend aussehen: "Unter den Studentinnen, die im 8.Monat schwanger sind und in einer Stadt mit mehr als 50000 Einwohnern wohnen, ist die Bereitschaft in der Vorweihnachtszeit eine Spende einer wohltätigen Organisation zu geben, wesentlich geringer als im gesamten Kundenstamm." Solche lokal beschreibende Aussagen sind möglicherweise eine wichtige Basis für die Planung von Geschäftsstrategien.[5]

[5] für eine klare Definition der Lernaufgaben siehe angegebenes Kapitel in Tabelle 2.1

2.7 Einordnung nachfolgender Abschnitte

In diesem Abschnitt sollte die Einordnung dieser Arbeit in die nachfolgenden Arbeiten erfolgen. Jedoch ist der Überblick über den Wissensentdeckungsprozess in Datenbanken der Einstieg, um die nachfolgenden Arbeiten in diesen Prozess einordnen zu können. Deshalb sollen hier vielmehr die nachfolgenden Arbeiten in den Wissensentdeckungsprozess eingeordent werden.

Die Merkmalsgenerierung und Merkmalsselektion (Kapitel 12) ist in Phase III der Datenvorbereitung einzuordnen, dafür ist jedoch ein mehr oder weniger großes Datenverständnis nötig. Dieser Abschnitt beschäftigt sich mit speziellen Techniken Selektion und Generierung von Daten. Wobei das Hintergrundwissen über die Daten in den Techniken eher vernachlässigt wird (Wrapper-Ansatz).

Es folgen weitere Abschnitte, die einen Überblick über verschiedene Data-Mining-Techniken geben, die in Phase IV zum Einsatz kommen können. Dabei werden Techniken vorgestellt, die aus verschiedenen Bereichen kommen. Techniken, die statistische, logische und numerische Grundideen verfolgen.
Dabei können die Verfahren auch für unterschiedliche Lernziele eingesetzt werden: Klassifikation, Regression, Subgruppenentdeckung, Entdeckung homogener Gruppen (Clustering). Zum Beispiel können Neuronale Netze für das überwachte Lernen sprich Klassifikation und Regression, sowie für das unüberwachte Lernen (Clustering) eingesetzt werden.

Eine Methode, die die Prädikatenlogik verwendet ist das RDT (Kapitel 6). Kapitel 10 beschäftigt sich mit dem Clustering. Nicht zu vergessen die Einführung in einen induktive Technik: Entscheidungsbaumlernverfahren (Kapitel 4).

Um sich mit den Daten in Phase II vertraut zu machen, ist auch Grundwissen über Verteilungen, Signifanztest und Sampling wichtig, welches im Abschnitt über das Handwerkszeug der Statistik geschieht (Kapitel 3). Dieser Abschnitt beschäftigt sich auch mit der Kreuzvalidierung, der zum besseren Training des Modells eingesetzt werden kann. Das Kapitel 13 gibt einen ersten Einblick in die Welt des KDD-Cups und macht an ausgewählten Beispielen deutlich in welcher Art und Weise die Phasen Datenverständnis, Datenvorbereitung, Data-Mining-Phase und Bewertung in den Siegerlösungen durchgeführt wurden.

Kapitel 3

Verteilungen, Signifikanztest, Kreuzvalidierung

Nazif Veliu

3.1 Einführung

Die beiden Disziplinen *Statistik* und *DataMining* haben insofern gemeinsame Ziele als sie sich beide mit der Entdeckung von Strukturen in Daten befassen. In der Tat überlappen sich ihre Ziele so sehr, dass einige Menschen (vor allem Statistiker) Data Mining als ein Untergebiet der Statistik ansehen. Das ist keine realistische Beurteilung. Data Mining gebraucht ebenso Ideen, Instrumente und Methoden aus anderen Gebieten - vor allem aus Gebieten wie Datenbankentechnologien und maschinellem Lernen - und befasst sich mit einigen Gebieten der Statistik nur am Rande [39].
Täglich werden Datenbanken von Unternehmen und Forschungseinrichtungen mit mehreren Terabyte an neuen Daten gefüllt (Supermarktketten, medizinische Untersuchungen, Reparaturdaten von Fahrzeugen, Satellitendaten, etc.). Ziel des Data Mining ist es, aus der Datenflut wertvolle Informationen und Zusammenhänge zu extrahieren und für unterschiedliche Zielgruppen aufzuarbeiten. Dabei kommen u.a. Methoden der Statistik und des maschinellen Lernens zum Einsatz. Für viele, die ironisch auf das explosionartig anwachsende kommerzielle Interesse im Bereich des Data Ming blicken, ist Data Mining gleichbedeutend mit Statistik und Marketing.
Stark vereinfacht kann man jedoch sagen, dass die Statistik mehr mit dem Überprüfen von Hypothesen zu tun hat und Data Mining Verfahren zur (automatischen) Mustererkennung in großen Datenbeständen zur Verfügung stellt [69]. Aufgabe der Statistik ist es auch, Rückschlüsse aus Beobachtungen zu ziehen, die unter dem Einfluss des Zufalls entstanden sind.

3.2 Grundlagen der Wahrscheinlichkeitsrechnung

3.2.1 Wahrscheinlichkeit

Endlicher Wahrscheinlichkeitsraum

Zunächst betrachten wir Zufallsexperimente mit endlich vielen möglichen Versuchsausgängen ω, die zusammengefasst die endliche Ergebnismenge Ω bilden. Wir identifizieren A $\subseteq$ von Ω mit dem Ereignis, dass ein $\omega \in A$ das beobachtete Ergebnis ist. A $\cap$ B ist das Ereignis, dass A und B eintreten, sowie A $\cup$ B, dass sich A oder B ereignen. Das Komplement A^C von A in Ω bezeichnet das Ereignis, dass A nicht geschieht. Die leere Menge $\emptyset$ heißt auch das unmögliche Ereignis; Ω das sichere Ereignis. Die Menge aller Ereignisse ist die Potenzmenge $P(\Omega)$.
Definition der Wahrscheinlichkeit bei einem Laplace-Experiment Bei einem Laplace-Experiment mit der Ergebnismenge $\Omega=\{\omega_1,\omega_2,...,\omega_n\}$ besitzen alle Elementarereignisse ω_i die gleiche Wahrscheinlichkeit

$$\mathrm{p}(\omega_i)=\frac{1}{m} \text{ (i=1,2,...,m)}$$

Die Wahrscheinlichkeit eines Ereignisses A ist dann durch die Formel
$\mathrm{p}(\omega_A)=\frac{g(A)}{m}$ gegeben.
Dabei bedeutet:

g(A): Anzahl der für das Ereignis A günstigsten Fälle (d.h. der Fälle, in denen das Ereignis A eintritt)

m: Anzahl der insgesamt möglichen Fälle
Man kann die Wahrscheinlichleit P(A) eines Ereignisses A auch in der Form
$\mathrm{P(A)}=\frac{\text{Anzahl g(A) der Elemente von A}}{\text{Anzahl m der Elemente von } \Omega}$
angeben.

Beispiel:
Beim Zufallsexperiment *Wurf eines homogenen Würfels* treten alle 6 möglichen Augenzahlen (Elementarereignisse) mit der gleichen Wahrscheinlichkeit auf:
$\mathrm{p(i)}=\frac{g(i)}{m}=\frac{1}{6}$ für i=1,2,...,6
Anzahl der günstigen Fälle: g(i)=1
Anzahl der möglichen Fälle: m=6

Für das Ereignis A=$\{2,4,6\}$ („gerade Augenzahl“) erhalten wir somit die Wahrscheinlichkeit

$$\mathrm{P(A)}=\frac{g(A)}{m}=\frac{3}{6}=\frac{1}{2}$$

Denn die Anzahl der für das Ereignis A günstigen Fälle ist g(A)=3, da A durch genau drei Elementarereignisse realisiert wird (A tritt ein bei der Augenzahl „2 “ oder „4 “ oder „6 “), während die Anzahl der möglichen Fälle m=6 beträgt (es gibt genau 6 Elementarereignisse) [57].

Wahrscheinlichkeitsaxiome

Relative Häufigkeiten

Die relativen Häufigkeiten von zufälligen Ereignissen, die bei einem genügend oft wiederholten Zufallsexperiment mit der Ergebnismenge Ω beobachtet werden, genügen den folgenden Gesetzmäßigkeiten und Regeln (n: Anzahl der Versuche):

1. Die relative Häufigkeit h_n(A) eines belibiegen Ereignisses A ist eine nichtnegative Zahl, die höchstens 1 sein kann. Es gilt also:

$$0 \leq h_n(\mathrm{A}) \leq 1$$

2. Für das sichere Ereignis Ω, das immer eintritt, gilt:

$$h_n(\Omega) = 1$$

3. Für zwei sich gegenseitig ausschließende Ereignisse A und B gilt stets:

$$h_n(\mathrm{A} \cup \mathrm{B}) = h_n(\mathrm{A}) + h_n(\mathrm{B})$$
$$\longrightarrow \text{Additiossatz}$$

4. **Erfahrungsregel**: Mit zunehmender Anzahl n der Versuche „stabilisiert“ sich im allgemeinen die relative Häufigkeit h_n(A) eines zufälligen Ereignisses A und schwankt somit immer weniger um einen bestimmten Wert h(A).

Wahrscheinlichkeitsaxiome von Kolmogoroff

Jedem Ereignis A eines Zufallsexperiments mit der Ergebnismenge Ω wird eine reelle Zahl P(A), Wahrscheinlichkeit des Ereignisses A genannt, so zugeordnet, daß die folgenden Axiome erfüllt sind.
A1: P(A) ist eine nicht negative Zahl, die höchstens gleich 1 ist:

$$0 \leq \mathrm{P}(\mathrm{A}) \leq 1$$

A2: Für das sichere Ereignis Ω gilt:

$$\mathrm{P}(\Omega) = 1$$

A3: Für paarweise sich gegenseitig ausschließende Ereignisse A_1,A_2,A_3,...,A_n gilt:

$$P(A_1 \cup A_2 \cup A_3 \cup ... \cup A_n)=P(A_1) + P(A_2) + P(A_3) + ... + P(A_n)$$

Aus diesen Axiomen lassen sich weitere Eigenschaften der Wahrscheinlichkeiten herleiten und zwar:

1. Für das unmögliche Ereignis $\emptyset$ gilt:
   $P(\emptyset) = 0$
2. Für das zum Ereignis A komplementäre Ereignis $\overline{A}$ gilt:
   $P(\overline{A})= 1 - P(A)$
3. Für zwei sich gegenseitig ausschließende Ereignisse A und B folgt aus A3:
   $P(A \cup B)= P(A) + P(B)$

Wahrscheinlichkeitsraum

Jedem Elementarereignis ω_i aus der Ereignismenge $\Omega=\{\omega_1,\omega_2,\omega_3,...\}$ eines Zufallsexperiments ordnen wir eine reelle Zahl p_i so zu, daß die beiden folgenden Bedingungen erfüllt sind:

1. $p_i \geq 0$ (i=1,2,...,m)
2. $\sum_{i=1}^{\infty} p_i=p_1 + p_2 + p_3 + ...= 1$

   Der Ereignisraum wird damit zu einem Wahrscheinlichkeitsraum.
   Die Wahrscheinlichkeits P(A) eines Ereignisses A aus dem Eriegnis- oder Wahscheinlichkeitsraum ist dann die Summe der Wahrscheinlichkeiten der in A enthaltenen Elemente (Elementarereignisse):
   $P(A) =\sum_i p_i$

Bedingte Wahrscheinlichkeit

In viele Anwendungen interessiert häufig die Wahrscheinlichkeit für das Eintreten eines bestimmten Ereignisses B unter der Voraussetzung oder Bedingung, dass ein anderes Ereignis A bereits eingetreten ist.
Die Wahscheinlichkeit für das Eintreten des Ereignisses B unter der Bedingung oder Voraussetzung, daß das Ereigniss A bereits eingetreten ist, heißt **bedingte Wahrscheinlichkeit** von B unter der Bedingung A und wird durch das Symbol P(B|A) gekennzeichnet. Sie wird durch die Gleichung

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \text{ mit } (P(A) \neq 0)$$

definiert. Oft ist in der Praxis die bedingte Wahrscheinlichkeit gegeben und die Schnittwahrscheinlichkeit gesucht. Dazu verwendet man die Gleichung für die bedingte Wahrscheinlichkeit und erhält so den **Multiplikationsatz**

$$P(A \cap B) = P(A) \cdot P(B|A), \text{ wenn } (P(A) \neq 0)$$

Mit der Formel der bedingten Wahrscheinlichkeit ist es plausibel, ein Ereignis B von einem Ereignis A unabhängig zu nennen, wenn $P(B)=P(B|A)$ gilt. Zwei Ereignisse A und B heißen **stochastisch unabhängig**, wenn

$$P(B|A) = P(B) \text{ bzw. } P(A \cap B) = P(A) \cdot P(B)$$

3.2.2 Zufallsgrößen

Die Ergebnisse von Zufallsexperimenten können qualitative oder quantitative Größen sein. Um qualitative Ergebnisse weiterverwerten und mit ihnen rechnen zu können, ist es sinvoll, sie durch quantitative Werte zu beschreiben. Man benötigt also eine Vorschrift, die jedem Ausgang eines Zufallsexperiments eine reelle Zahl zuordnet.

Definition:
Unter einer Zufallsgröße oder Zufallsvariable X verstehen wir eine Funktion, die jedem Elementarereignis ω aus der Ergebnismenge Ω eines Zufallsexperiments genau eine reelle Zahl $X(\omega)$ zuordnet.
Sie werden üblicherweise mit großen lateinischen Buchstaben gekennzeichnet. Man unterscheidet noch zwischen einer diskreten und einer stetigen Zufallsvariable.
Eine Zufallsvariable X heißt dabei diskret, wenn sie nur endlich viele Werte annehmen kann.
Eine Zufallsvariable X heißt dagegen stetig, wenn sie jeden beliebigen Wert aus einem (reellen) endlichen oder unendlichen Intervall annehmen kann.

Beispiel:
Ein homogener Würfel wird *fünfmal* geworfen und dabei wird festgestellt, wie oft die Augenzahl „1 “ auftritt. Dann ist die Größe *X= Anzahl von Würfen mit der Augenzahl* „1“ eine diskrete Zufallsvariable mit den möglichen Werten 0,1,2,3,4 und 5.

Verteilungsfunkion einer Zufallsvariable

Die Verteilungsfunktion F(x) einer Zufallsvariable X ist die Wahrscheinlichkeit dafür, daß die Zufallsvariable X einen Wert annimmt, der kleiner oder gleich einer vorgegebenen reellen Zahl x ist:

$$F(x) = P(X \leqslant x)$$

Eine Zufallsvariable wird durch die Verteilungsfunktion F(x) vollständig beschrieben. Verteilungsfunktionen besitzen ganz allgemein die folgenden Eigenschaften:

1. F(x) ist eine monoton wachsende Funktion mit $0 \leqslant F(x) \leqslant 1$.
2. $\lim_{x \to -\infty} F(x) = 0$ (unmögliches Ereignis)
3. $\lim_{x \to \infty} F(x) = 1$ (sicheres Ereignis)
4. $P(a < X \leqslant b) = F(b) - F(a)$

Wahrscheinlichkeitsverteilung einer diskreten Zufallsvariable

Die Wahrscheinlichkeitsverteilung einer diskreten Zufallsvariablen X läßt sich durch die Wahrscheinlichkeitsfunktion

$f(x) = p_i$ für $x = x_i$ (i=1,2,3...) und

$f(x) = 0$ für alle übrigen x
oder durch die zugehörige Verteilungsfunktion
$F(x) = P(X \leqslant x) = \sum f(x_i)$ über alle $x_i \leqslant x$ vollständig beschreiben.
p_i: Wahrscheinlichkeit dafür, daß die Zufallsvariable X den Wert x_i annimmt

Wahrscheinlichkeitsfunktion f(x) und Verteilungsfunktion F(x) besitzen folgende Eigenschaften:

1. $f(x_i) \geq 0$
2. f(x) ist normiert, d.h. es gilt
   $\sum_{i=1}^{\infty} f(x_i) = 1$
3. F(x) ist eine monoton wachsende Funktion mit $0 \leqslant F(x) \leqslant 1$.
4. $P(a < X \leqslant b) = F(b) - F(a)$

Wahrscheinlichkeitsverteilung einer stetigen Zufallsvariable

Die Wahrscheinlichkeitsverteilung einer stetigen Zufallsvariablen X läßt sich durch die Wahrscheinlichkeitsdichtefunktion f(x) oder durch die zugehörige Verteilungsfunktion

$$F(x) = P(X \leqslant x) = \int_{-\infty}^{x} f(u)du$$

vollständig beschreiben.
Die Dichtefunktion f(x) und Verteilungsfunktion F(x) besitzen dabei die folgenden Eigenschaften:

1. f(x)$\geq$ 0
2. f(x) ist normiert, d.h. es gilt
   $\int_{-\infty}^{\infty}$ f(x)=1
3. Die monoton wachsende Verteilungsfunktion F(x) ist eine Stammfunktion der Dichtefunktion f(x), d.h. es gilt F'(x)=f(x)
4. Die Wahrscheinlichkeit dafür, daß die stetige Zufallsvariable X einen Wert zwischen a und b annimmt, berechnet sich dann wie folgt:
   P(a<X$\leqslant$b)= $\int_a^b$ f(x)dx=F(b) - F(a),

3.2.3 Parameter von Wahrscheinlichkeitsverteilung

Die Wahrscheinlichkeitsverteilung einer (diskreten oder stetigen) Zufallsvariable X läßt sich in eindeutiger Weise entweder durch die Verteilungsfunktion F(x) oder aber durch die zugehörige Wahrscheinlichkeits- bzw. Dichtefunktion f(x) beschreiben. Die Verteilung kann aber auch durch bestimmte Parameter, die man als Kennwerte der Verteilung bezeichnet, charakterisiert werden. Zu ihnen zählen u.a
- der *Mittel- oder Erwartungswert* μ,
- die *Varianz* σ^2 und
- die *Standardabweichung* σ

Erwartungswert einer Zufallsvariable

Definition:
Unter dem Erwartungswert E(X) einer diskreten Zufallsvariable X mit der Wahrscheinlichkeitsfunktion f(x) versteht man die Größe

$$E(X)=\sum_i x_i \cdot f(x_i)$$

Definition:
Unter dem Erwartungswert E(X) einer stetigen Zufallsvariable X mit der Dichtefunktion f(x) versteht man die Größe

$$E(X)=\int_{-\infty}^{\infty} x \cdot f(x)$$

Mittelwert, Varianz und Standardabweichung einer Zufallsvariable

Im Falle einer diskreten Zufallsvariable X lautet die Definition der drei Kennwerte *Mittelwert* μ, *Varianz* σ^2 *und Standardabweichnung* σ wie folgt:

Definition:
Der diskreten Zufallsvariable X mit der Wahrscheinlichkeitsfunktion f(x) werden die folgenden Kennwerte zugeordnet:

1. **Mittelwert** μ

$$\mu = \mathrm{E(X)} = \sum_i x_i \cdot \mathrm{f}(x_i)$$

2. **Varianz** σ^2

$$\sigma^2 = \mathrm{Var(X)} = \sum_i (x_i - \mu)^2 \cdot \mathrm{f}(x_i)$$

3. **Standardabweichung** σ

Die Standardabweichung σ ist die Quadratwurzel aus der Varianz

$$\sigma^2 = \mathrm{Var(X)}$$

$$\sigma = \sqrt{Var(X)}$$

Bei einer stetigen Zufallsvariable X werden die drei Kennwerte Mittelwert μ, Varianz σ^2 und Standardabweichung σ wie folgt in der Integralform definiert:

Definition :
Der stetigen Zufallsvariable X mit der Dichtefunktionfunktion f(x) werden die folgenden Kennwerte zugeordnet:

1. **Mittelwert** μ

$$\mu = \mathrm{E(X)} = \int_{-\infty}^{\infty} \mathrm{x} \cdot \mathrm{f(x)}$$

2. **Varianz** σ^2

$$\sigma^2 = \mathrm{Var(X)} = \int_{-\infty}^{\infty} (\mathrm{x} - \mu)^2 \cdot \mathrm{f(x)dx}$$

3. **Standardabweichung** σ

Die Standardabweichung σ ist die Quadratwurzel aus der Varianz $\sigma^2 = \mathrm{Var(X)}$:

$$\sigma = \sqrt{Var(X)}$$

3.3 Spezielle Wahrscheinlichkeitsverteilungen

3.3.1 Binomialverteilung

Zufallsexperimente mit nur zwei verschiedenen Ausgängen (Ergebnissen) führen zur Binomialverteilung. Bei einem solchen Experiment tritt ein Ereignis A mit der Wahrscheinlichkeit p und das zu A komplementäre $\bar{A}$ mit der Wahrscheinlichkeit q=1-p ein.
Ein Zufallsexperiment mit den beiden sich gegenseitig ausschließenden Ereignissen A und $\bar{A}$ werde n-mal ausgeführt, wobei das Ereigniss A bei jeder Durchführung des Experiments mit der gleichen und somit konstanten Wahrscheinlichkeit p eintritt. Dann genügt die diskrete Zufallsvariable X= *Anzahl der Versuche, in denen das Ereignis A eintritt* der sog. Binomialverteilung mit der Wahrscheinlichkeitsfunktion

$$f(x) = P(X = x) = \binom{n}{x} \cdot p^x \cdot q^{n-x}, (x = 0, 1, 2, ..., n)$$

und der dazugehörigen Verteilungsfunktion

$$F(X) = P(X \leq x) = \sum_{k \leq x} \binom{n}{k} \cdot p^k \cdot q^{n-k}, (x \geq 0)$$

Abb.1 zeigt den Verlauf der Wahrscheinlichleitsfunktion f(x) für n=20 und p=0.5:

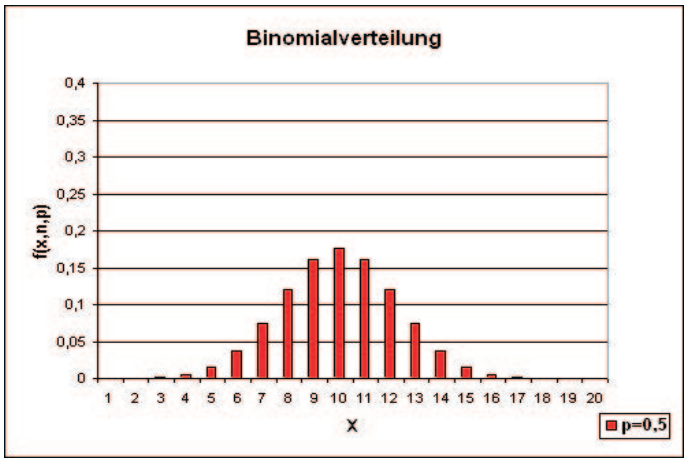


Abb.1
Wahrscheinlichkeitsfunktion f(x) einer Binomialverteilung für den Parameter n=20 und p=0.5

Die Kennwerte dieser Verteilung lauten:
- Mittelwert: $\mu = np$
- Varianz: $\sigma^2 = npq = np(1-p)$
- Standardabweichung: $\sigma = \sqrt{npq}$

Dabei bedeuten:
p: konstante Wahrscheinlichkeit für das Eintreten des Ereignisses A beim Einzelversuch
q: Konstante Wahrscheinlichkeit für das Eintreten des zu A komplementären Ereignisses $\bar{A}$ beim Einzelversuch
n: Anzahl der Ausführungen

3.3.2 Poisson-Verteilung

Die Poissonverteilung beschreibt das Auftreten von sehr seltenen Ereignissen in einem Zeit-Raum Kontinuum. Sie dient auch der Beschreibung der Anzahl von Ereignissen je Betrachtungseinheit.
Beispiele:
Bestellungen je Kunde, Reklamation je Kunde, Buchungen je Quartal, Anfragen je Woche, Telefonanrufe je Stunde, Verkehrsunfälle je Woche.
Solche Ereignisse genügen der Poisson-Verteilung mit der Wahrscheinlichkeitsfunktion

$$f(x) = P(X = x) = \frac{\lambda^x}{x!} \cdot e^{-\lambda}, (x = 0, 1, 2, ...)$$

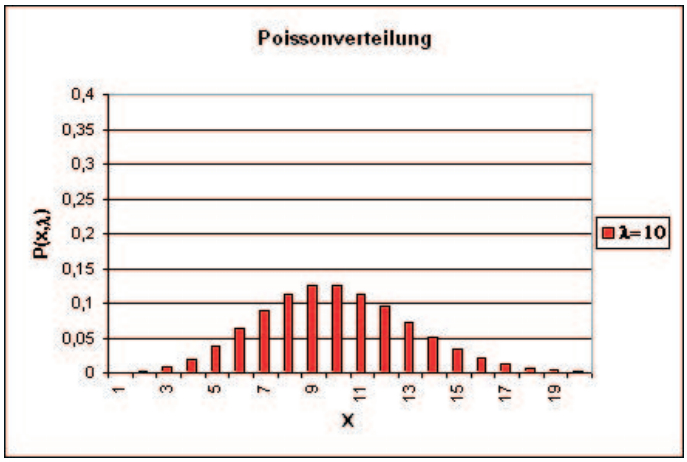


Abb. 2
Wahrscheinlichkeitsfunktion f(x) einer Poisson-Verteilung mit dem Parameter $\lambda = 10$

Der in der Verteilung auftretende positive Parameter λ ist zugleich der Erwartungwert oder Mittelwert der Verteilung: E(X)=λ. Die Varianz Var(X)=$\sigma^2 = \lambda$ stimmt mit dem Mittelwert stets überein.
Die Verteilunsfunktion der Poisson-Verteilung lautet:

$$F(x) = P(X \leq x) = e^{-\lambda} \cdot \sum_{k \leq x} \frac{\lambda^k}{k!}$$

Beispiel:

An einer Kreuzung findet im Mittel 2 Unfälle pro Woche statt, d.h. $\lambda = 2$.
Wir wollen die Wahrscheinlichkeit dafür berechnen, daß in einer Woche „kein Unfall " stattfindet und die Wahrscheinlichkeit, daß „weniger als 3 Unfälle " in einer Woche stattfinden.
Lösung:

X= Anzahl der Unfälle
Die Wahrscheinlichkeit, daß in einer Woche „kein Unfall" stattfindet ist
$P(X = x) = \frac{\lambda^x}{x!} \cdot e^{-\lambda} = P(X = 0) = \frac{2^0}{0!} \cdot e^{-2} \approx 0,14$

Abb. 3

Die Wahrscheinlichkeit, daß in einer Woche „weniger als 3 Unfälle " stattfinden ist
$P(X \leq 2) = P(X = 0) + P(X = 1) + P(X = 2) \approx 0,14 + 0,27 + 0,27 \approx 0,68$

Abb. 4

3.3.3 Exponentialverteilung

Zufallsvariable, bei denen die Zeit eine entscheidene Rolle spielt sind häufig exponentialverteilt.
Beispiel dafür sind: Dauer von Telefongesprächen, Lebensdauer des radioaktiven Zerfalls, Arbeitszeit einer Maschine zwischen zwei Stillständen, Lebensdauer von Bauteilen oder Lebewesen, Ankunftszeiten in einem Bedienungsschalter

etc.
Die Verteilung einer solchen Zufallsvariable mit der Dichtefunktion

$$f(x) = \lambda e^{-\lambda x} : x \geq 0 \text{ und}$$
$$f(x) = 0 : x < 0$$

und der Verteilungsfunktion

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \int_{-\infty}^{x} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dt,$$

heißt Exponentialverteilung.

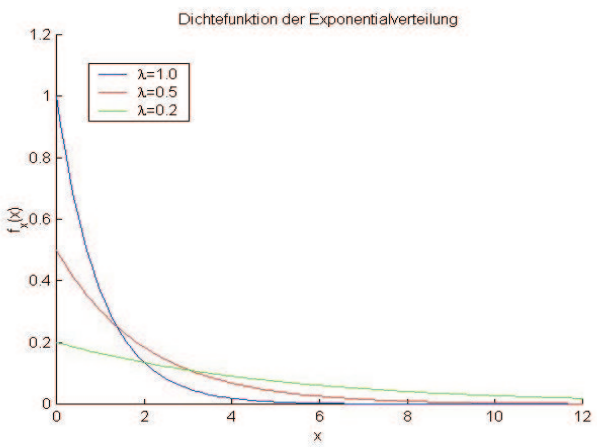


Abb.5
Die Dichtefunktion f(x) der Exponentiallvertelung

3.3.4 Gaußsche Normalverteilung

Die Verteilung einer stetigen Zufallsvariable X mit der Dichtefunktion

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}, (-\infty < x < \infty)$$

und der Verteilungsfunktion

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \int_{-\infty}^{x} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dt,$$

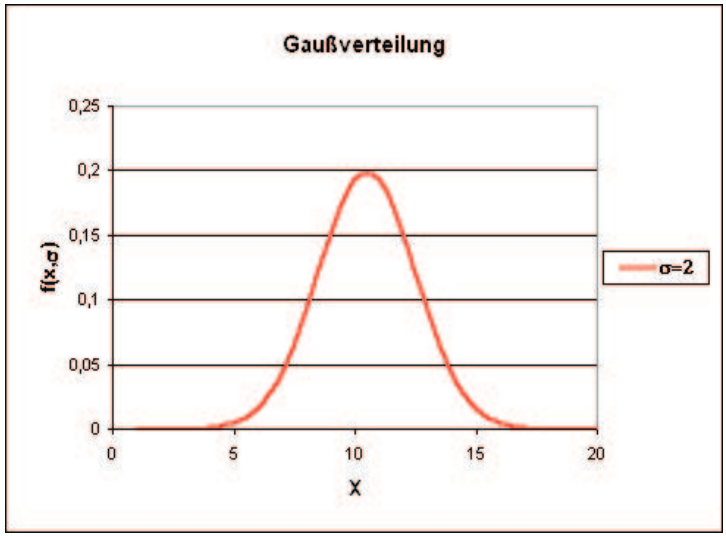


Abb. 6
Dichtefunktion f(x) der Gaußschen Normalverteilung

heißt Gaußsche Normalverteilung mit den Parameter μ und $\sigma > 0$, die zugleich spezielle Kennwerte dieser Normalverteilung sind:
- μ: ist der Mittelwert-oder Erwartungwert,
- σ: die Standardabweichung und
- σ^2: die Varianz
Während der erste Parameter μ die Lage des Maximums festlegt, bestimmt der zweite Parameter σ Breite und Höhe der Glockenkurve.

3.3.5 Standardnormalverteilung

Die Gaußsche Normalverteilung mit dem Parameter μ (Mittelwert) und σ (Standardabweichung) lässt sich stets auf die sog. Standardnormalverteilung mit den speziellen Parameterwerten $\mu = 0$ und $\sigma = 1$ zurückführen. Alle Werte die aus einer Normalverteilung kommen, müssen in „z-Werte“ konvertiert werden. Der Übergang erfolgt mit Hilfe der linearen Transformation:

$$Z = \frac{X-\mu}{\sigma}$$

Die Dichtefunktion der standardisierten Normalverteilung lautet somit:

$$\varphi(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{1}{2}z^2}$$

und die zugehörige Verteilungsfunktion ist:

$$\phi(z) = P(Z \leq z) = \frac{1}{\sqrt{2\pi}} \cdot \int_{-\infty}^{z} e^{-\frac{1}{2}t^2} dt$$

3.3.6 Testverteilungen

t-Verteilung

Die t-Verteilung von Student bildet die Grundlage für bestimmte Parametertest in der Statistik. Sie ist ähnlich wie die z-Verteilung (Standardnormalverteilung), wobei X eine normalverteilte Zufallsvariable und S(x) jedoch eine Abschätzung der Standardabweichung (Varianz der Stichprobe) von X von einer Stichprobe aus X darstellt und nicht die Grundgesamtheit-Standardabweichung σ. Grundsätzlich wurde die t-Verteilung ausgearbeitet um ein adäquates Tool für kleine Stichprobenumfänge zu erhalten. Dabei ist auch die Anzahl der Freiheitsgrade zu berücksichtigen, wobei bei einer infiniten Anzahl an Freiheitsgraden, die t-Verteilung dasselbe wie die z-Verteilung ist, da die Abschätzung der Standardabweichung S(x) sehr genau und annäherend gleich σ ist.

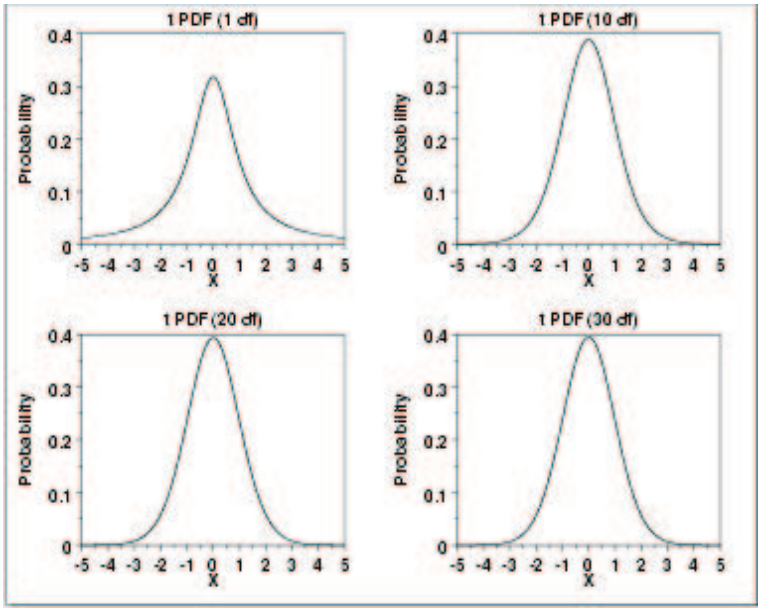


Abb. 7
Dichtefunktion der t-Verteilung

Chi-Quadrat-Verteilung

Die χ^2-Verteilung (Chi-Quadrat-Verteilung) stellt die Grundlage des sog. χ^2-Tests für Hypothesen mit nominalskalierten Daten dar. Die Signifikanztests mit Hilfe der Verteilung sind nicht parametrische Tests. Durch χ^2-Verfahren kann außerdem geprüft werden, ob eine empirische Häufigkeitsverteilung normalverteilt ist. Die zugehörige Verteilungsfunktion $F(\chi^2)$ ist ähnlich wie die

Standardnormalverteilung.
Herleitung:
Das Quadrat einer standardnormalverteilten Zufallsvariable Z wird als χ^2-verteilte Zufallsvariable bezeichnet: $\chi^2 = z^2$. Zieht man n voneinander unabhängige, standardnormalverteilte Zufallsvariablen $Z_1, Z_2 ... Z_n$ und addiert deren Quadrate, so erhält man eine Summe die als χ^2_n bezeichnet wird.

$$\chi^2_n = \sum_{i=0}^{n} z_i^2$$

In Abhängigkeit von der Anzahl n der z^2-Variablen sprechen wir von χ^2-Verteilungen mit unterschiedlichen Freiheitsgraden (FG oder df degrees of freedom). Auch bei den χ^2-Verteilungen handelt es sich also um eine Klasse unterschiedlicher Verteilungen.
Eine χ^2-Verteilung hat folgende Eigenschaften:
- $\mu = df$ und $\sigma = 2df$
- Für eine große Anzahl von Freiheitgraden df nähert sich die χ^2-Verteilung einer Normalverteilung N (df,$\sqrt{2df}$) an.

3.3.7 Konfidenzintervalle

Der aus einer Zufallsstichprobe gewonnenen Schätzwert für einen Parameter kann noch erheblich vom tatsächlichen (aber unbekannten) Wert abweichen, insbesondere bei einem kleinen Stichprobenumfang. Aber *wie genau* und *wie sicher* sind die aus Stichprobenuntersuchung mit Hilfe von Schätzfunktionen bestimmten Näherungs- oder Schätzwerte für die unbekannten Parameter ?
Die Antwort auf diese Frage liefert das Konfidenzintevall.

Konfidenzintervall für den unbekannten Parameter ϑ einer von Typ her bekannten Wahrscheinlichkeitsverteilung
Es sei eine Zufallsvariable X gegeben, deren Wahrscheinlichkeitsverteilung (Verteilungsfunktion) noch einen unbekannten Parameter ϑ enthalte. Für diesen Parameter lässt sich dann unter Verwendung einer konkreten Zufallsstichprobe $x_1, x_2, ..., x_n$ wie folgt schrittweise ein sog. Vertrauens- oder Konfidenzintervall bestimmen:

1. Man wählt zuerst ein bestimmtes Vertrauensniveau
   $\gamma = 1 - \alpha (0 < \gamma < 1)$.
   α ist dabei die sog. Irrtumswahrscheinlichkeit.

2. Man bestimmt zwei Zufallsgrößen Θ_u und Θ_o so, daß sie mit einer beliebig gewählten, aber großen Wahrscheinlichkeit γ Werte annehmen, die den wahren, aber unbekannten Parameterwert ϑ einschließen. Somit gilt

   $P(\Theta_u \leq \vartheta \leq \Theta_o) = \gamma$

   Die Werte der beiden Zufallsvariablen Θ_u und Θ_o müssen dabei aus den

Stichprobenwerten $x_1, x_2, ..., x_n$ einer vorgegebenen konkreten Stichprobe berechnen lassen und variieren daher von Stichprobe zu Stichprobe. Die Zufallsvariablen sind demnach Stichprobenfunktionen der *n unabhängigen* Zufallsvariablen $X_1, X_2, ..., X_n$, die alle die gleiche Verteilungsfunktion besitzen wie die Zufallsvariable X.
Die zwei Stichprobenfunktionen
$\Theta_u = g_u(X_1; X_2; ...; X_n)$ und
$\Theta_o = g_o(X_1; X_2; ...; X_n)$
schließen den wahren Wert des Parameters ϑ mit der gewählten Wahrscheinlichkeit $\gamma = 1 - \alpha$.
$P(\Theta_u \leq \vartheta \leq \Theta_o) = \gamma = 1 - \alpha$.

3. Aus der vorgegeben konkreten Stichproben $x_1, x_2, ..., x_n$werden die Werte der beiden Stichprobenfunktionen Θ_u und Θ_o berechnet:
   $c_u = g_u(x_1; x_2; ...; x_n)$
   $c_o = g_o(x_1; x_2; ...; x_n)$
   Sie liefern uns die Grenzen des gesuchten Vertrauens- oder Konfidenzintervalls.

4. Das Vertrauens- oder Konfidenzintervall für den unbekannten Parameter ϑ lautet damit wie folgt:
   $c_u \leq \vartheta \leq c_o$
   Der wahre Wert des Parameters ϑ liegt dann mit dem Vertrauen von $\gamma \cdot 100\%$ in diesem Intervall.

Beispiel:

Angenommen, wir messen die Erfolgsrate eines Klassifizierers anhand einer Testmenge (n=40) und erhalten einen numerischen Wert von 30%. D.h, daß die Anzahl der Erfolge aus dieser Stichprobe $k = 12$ beträgt
Nun ist dies nur eine Schätzung. Was kann man über die echte Erfolgsrate in der Grundgesamtheit sagen? Man erwartet einen Wert nahe an 30%. Aber wie nah?
Hier handelt es sich um eine Folge von unabhängigen Ereignisse, die erfolgreich sind oder fehlschlagen können. Sie wird in der Statistik auch als Bernoulli-Prozess bezeichnet.

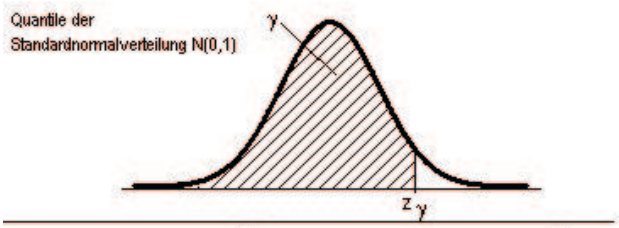
Gesucht ist also die wahre Erfolgsrate p

Lösung:
Der Mittelwert und die Varianz eines einzigen Bernoulli-Versuchs mit einer wahren Erfolgsrate p ist p bzw. p(1-p). Werden n Versuche aus einem Bernoulli-Prozess genommen, ist die erwartete Erfolgsrate $\hat{p} = \frac{k}{n}$ eine Zufallsvariable mit

demselben Mittelwert p. Wenn n groß genug ist, nähert sich die Verteilung dieser Zufallsvariable der Normalverteilung an.

1. Wir wählen ein bestimmtes Vertrauensniveau $\gamma = 0.95\%$

2. Berechnung der Konstanten c aus der Bedingung:



	γ	z_γ		γ	z_γ
	0,965	1,8119		0,83	0,9542
	0,960	1,7507		0,82	0,9154
	0,955	1,6954		0,81	0,8779
Bsp.→	**0,950**	**1,6449**	←	0,80	0,8416
	0,945	1,5982		0,79	0,8064
	0.975	**1,960**			

Abb. 7

$P(-c \leq Z \leq c) = \gamma = 0.95$
$P(-c \leq Z \leq c) = 2\phi(c) - 1 = 0.95$
$\phi(c) = 0.975 \longrightarrow c = z_{0.975} = 1.960$
Die gesuchte Konstante ist demnach das Quantil $z_{0.975} = 1.960$ der Standardnormalverteilung.

3. Mit n=40 und k=12 ergibt sich für den unbekannten Parameter p folgender Schätzwert (beobachteter Wert): $\hat{p} = \frac{k}{n} = \frac{12}{40} = 0.30 = 30\%$

4. Das Vertrauens- oder Konfidenzintervall für den unbekannten Parameter p lautet somit:

$$\hat{p} - \frac{c}{n}\sqrt{n\hat{p}(1-\hat{p})} \leq p \leq \hat{p} + \frac{c}{n}\sqrt{n\hat{p}(1-\hat{p})} =$$

$$0.30 - \frac{1.960}{40}\sqrt{40 \cdot 0.30(1-0.30)} \leq p \leq 0.30 + \frac{1.960}{40}\sqrt{40 \cdot 0.30(1-0.30)} =$$

$$0.30 - 1.960 \cdot 0.07 \leq p \leq 0.30 + 1.960 \cdot 0.07 =$$

$$0.16 \leq p \leq 0.43$$

Es stellt sich heraus, daß die wahre Erfolgsrate p bei einer Wahrscheinlichkeit von etwa 95 % zwischen 16% und 43% liegt.

3.4 Signifikanztest

Unter einer statistische Hypothese (kurz: Hypothese) versteht man irgenwelche Annahmen, Vermutungen oder Behauptungen über die Wahrscheinlichkeitsverteilung einer Zufallsvariable oder Grundgesamtheit und über deren Parameter.
Ein Parametertest ist ein statistisches Prüfverfahren für einen unbekannten Parameter in der Wahrscheinlichkeitsverteilung einer Zufallsvariablen oder Grundgesamtheit, wobei die Art der Verteilung (d.h der Verteilungstyp wie z.B Binomialverteilung oder Gaußsche Normalverteilung) als bekannt vorausgesetzt wird. Ein solcher Test dient der Überprüfung einer Hypothese über einen bestimmten Parameter der Verteilung mit Hilfe einer Stichprobenuntersuchung der betreffenden Grundgesamtheit. Die zu überprüfende Hypothese wird meist als Nullhypothese H_0 bezeichnet. Ihr wird eine Alternativhypothese H_1 gegenübergestellt.
Das Ziel eines Parametertests ist dann, eine Entscheidung darüber zu ermöglichen, ob die Nullhypothese H_0 angenommen werden kann oder ob man sie zugunsten der Alternativhypothese H_1 verwerfen muß.

Über Planung und Durchführung eines Parametertests

Die Wahrscheinlichkeitsverteilung einer Zufallsvariable X sei zwar von der Art her bekannt, enthalte jedoch einen oder sogar mehrere unbekannte Parameter. So hat man es in den meisten Anwendungen z.B. häufig mit Normalverteilungen zu tun, deren Parameter μ und σ bzw. σ^2 jedoch unbekannt sind. Ein Parametertest für den unbekannten Parameter ϑ einer Wahrscheinlichkeitsverteilung läßt sich dann wie folgt planen und durchführen:

1. Zunächst formulieren wir Nullhypothese H_0 und Alternativhypothese H_1 anhand der vorgegebenen konkreten Fragestellung. Zum Beispiel testen wir die
   Nullhypothese $H_0 : \vartheta = \vartheta_0$ (der Parameter ϑ besitzt den Wert ϑ_0) gegen die
   Alternativhypothese $H_1 : \vartheta \neq \vartheta_0$.

2. Wir wählen dann eine bestimmte Signifikanzzahl α, häufig auch Signifikanzniveau genannt ($0 < \alpha < 1$). Sie ist die Wahrscheinlichkeit dafür, dass die Nullhypothese abgelehnt wird, obwohl sie richtig ist und heißt daher auch Irrtumswahrscheinlichkeit. In der Praxis wird α daher klein gewählt, übliche Werte sind $\alpha = 0.05 = 5\%$ oder $\alpha = 0.01 = 1\%$.

3. Für die Durchführung des Parametertests wird eine geeignete Test- oder Prüfvariable T bestimmt, die noch von den n unabhängigen Zufallsvariablen $X_1; X_2; ...; X_n$ abhängt, die alle die gleiche Verteilung besitzen wie

die ZV X:
$T = g(X_1; X_2; ...; X_n)$.

4. Dann bestimmen wir auf der Basis der gewählten Signifikanzzahl α zwei sog. kritische Grenzen c_u und c_o, derart, daß die Testvariable T mit der Wahrscheinlichkeit $\gamma = 1 - \alpha$ Werte aus dem Intervall $c_u \leq T \leq c_o$ annimmt.
   Bestimmungsgleichung für die kritischen Grenzen lautet somit:
   $P(c_u \leq T \leq c_o)_{H_0} = \gamma = 1 - \alpha$.

5. Wir berechnen jetzt den Wert der Testvariable T aus einer vorgegebenen konkreten Stichprobe vom Umgang n.
   $\hat{t} = g(x_1; x_2; ...; x_n)$.

6. **Testentscheidung:** Wir sind jetzt in der Lage, eine Entscheidung über Annahme oder Ablehnung der Nullhypothese H_0 zu treffen.
   1. Fall: Der Testwert- oder Prüfwert $\hat{t}$ fällt in den nichtkritischen Bereich der Tastvariable T, d.h. es gilt $c_u \leq \hat{t} \leq c_o$. Dann wir die Nullhypothese angenommen. Man bezeichnet diesen Bereich auch als Annahmebereich.

   2. Fall: Der Testwert- oder Prüfwert $\hat{t}$ fällt in den kritischen Bereich. Die Nullhypothese H_0 ist dann nicht haltbar, wir müssen sie zugunsten der Alternativhypothese H_1 ablehnen.

Beispiel:(z-Test)
Gegeben ist eine normalverteilte Stichprobe mit unbekanntem Mittelwert μ einer Normalverteilung bei bekannter Varianz σ^2
Stichprobenumfang: n=10;
Signifikanzniveau: $\alpha = 0.01$
Varianz: $\sigma^2 = 9$
Stichprobenmittelwert: $\bar{x} = 20$

Gesucht: Hypothesenüberprüfung über den Erwartungswert μ ?

Lösung:

Wir testen die Nullhypothese
$H_0 : \mu = \mu_0 = 22$
gegen die Alternativhypothese
$H_0 : \mu \neq 22$ schrittweise wie folgt:

1. Die Signifikanzzahl oder Irrtumswahrscheinlcihkeit α ist bereits vorgegeben: $\alpha = 0.01$

2. Die Bestimmungsgleichung für den kritischen Wert c der normalverteilten Testvariablen

$Z = \frac{\bar{X}-\mu_0}{\frac{\sigma}{\sqrt{n}}} = \frac{\bar{X}-22}{\frac{3}{\sqrt{10}}}$
lautet:
$P(-c \leq Z \leq c)_{H_o} = 1 - \alpha = 1 - 0.01 = 0.99$
Hieraus erhalten wir unter der Verwendung der Tabelle von Quantile der Standardnormalverteilung den folgenden Wert für die unbekannte Schranke c:
$P(-c \leq Z \leq c)_{H_o} = \phi(c) - \phi(-c) = \phi(c) - [1 - \phi(c)] = 2\phi(c) - 1 = 0.99\phi(c) = 0.995 \longrightarrow c = z_{0.995} = 2.576$
Der Annahmebereich wird somit durch das symetrische Intervall $-2.576 \leq z \leq 2.576$ beschrieben.

3. Der Stichprobenmittlewert $\bar{x}$ beträgt 20.
   Die Testvariable Z besitzt demnach den folgenden Testwert:
   $\bar{z} = \frac{\bar{x}-\mu_0}{\frac{\sigma}{\sqrt{n}}} = \frac{20-22}{\frac{3}{\sqrt{10}}} = -2.108$

4. Testentscheidung: der Testwert $\hat{z} = -2.108$ fällt in den Annahmebereich, d.h es gilt $2.576 \leq \hat{z} \leq 2.576$
   Die Nullhypothese kann daher aufgrund der verwendeten Stichprobe nicht abgelehnt werden. Wir können davon ausgehen, daß die normalverteilte Grundgesamtheit, aus der wir die Stichprobe entnommen haben, den Mittelwert $\mu_0 = 22$ besitzt.

   Anmerkung
   Für die Zufallsvariable X erhalten wir aus:
   $-c \leq \frac{\bar{x}-\mu}{\frac{\sigma}{\sqrt{n}}} \leq c \Leftrightarrow -\frac{c\sigma}{\sqrt{n}} \leq \bar{x} - \mu \leq \frac{c\sigma}{\sqrt{n}} = -2.44 \leq \bar{x} - \mu \leq 2.44$
   $\Rightarrow (22 - 2.44) \leq \bar{x} \leq (22 + 2.44)$
   $\Rightarrow (19.56) \leq \bar{x} \leq (24.44)$

   Der beobachtete Stichprobenmittelwert $\bar{x}$=20 liegt in diesem Intervall.

3.5 Bewertungsmaße

Für die Bewertung von KDD-Algorithmen verfährt man im Prinzip ähnlich wie bei der Bewertung von Information Retrieval Verfahren. Man verwendet ein Testset, das möglichst repräsentativ für die Verteilung der Beispiele in dem Bereich sein sollte, der mit einem Algorithmus bearbeitet werden soll. Aufgrund der richtig bzw. falsch kategorisierten Beispiele des Testsets kann man die Precision -und Recallwerte für jede Kategorie berechnen.

3.5.1 Accuracy

Die Accuracy mißt die Wahrscheinlichkeit, daß ein von einer Hypothese klassifizierendes Beispiel korrekt klassifiziert wurde.

Für ein *2-Klassen* Problem gilt:

Aktuell/Vorhersage	*Klasse 1*	*Klasse 2*
Klasse 1	a	c
Klasse 2	b	d

a, d : Anzahl der Instanzen, die als richtig klassifiziert wurden
b, c : Anzahl der Instanzen, die als falsch klassifiziert wurden

$$Accuracy: \ a := \frac{a+d}{a+b+c+d}$$

3.5.2 Precision

Die Precision gibt den Anteil der relevanten an der gefundenen Dokumenten wieder [32].

$$\text{Precision: } p := \frac{|REL \cap GEF|}{|GEF|}$$

REL: die Menge der relevanten Objekte in der Datenbank
GEF: die Menge der gefundenen Antwortobjekte

Analog dazu kann man in DataMining die Precision folgendermaßen definieren:
Die Precision gibt die Anzahl der Instanzen aus einer Klasse an, die richtig vorhergesagt wurden.

$$p_1 := \frac{a}{a+b} \text{ und } p_2 := \frac{c}{c+d}$$

a, d: Anzahl der Instanzen, die als richtig klassifiziert wurden
b, c: Anzahl der Instanzen, die als falsch klassifiziert wurden

3.5.3 Recall

Recall dagegen bezeichnet den Anteil der relevanten Dokumente, die tatsächlich gefunden wurden.

$$\text{Recall: } r := \frac{|REL \cap GEF|}{|REL|}$$

Analog dazu kann man auch Recall in DataMining so definieren:
Recall gibt die Anzahl der einer Klasse zugeordneten Instanzen an, die richtig klassifiziert wurden.

$$r_1 := \frac{a}{a+c} \text{ und } r_2 := \frac{b}{b+d}$$

a, d: Anzahl der Instanzen, die als richtig klassifiziert wurden
b, c: Anzahl der Instanzen, die als falsch klassifiziert wurden
Da es sich hier um stochastische Experimente handelt, sollte man die Messwerte auch entsprechend interpretieren. Im Falle der Precision p wird damit die Wahrscheinlichkeit approximiert, daß ein (zufällig ausgewähltes) gefundenes Dokument relevant ist.
Analog schätzt man mit dem Recall r die Wahrscheinlichkeit, daß ein relevantes Dokument gefunden wird.
Wenn man z.B. zwei (Klassifizierungs)Verfahren vergleichen will, um festzustellen, welche der Verfahren besser ist, dann sollte man nach einem hohem Precision und Recall-Maß streben.
Häufig ist es so, daß ein Verfahren ein höheren Recall-Wert und das andere einen höheren Precision-Wert hat. Dann ist es nicht mehr möglich, eine Aussage bezüglich einer Überlegenheit eines der beiden Verfahren zu machen.

3.5.4 F-Maß

Das F-Maß ist eine gängige Methode, (r,p)-Paare durch eine einzige Zahl auszudrücken. Abhängig von einem zu wählenden Parameter β berechnet sich dieses Maß zu

$$F_\beta = \frac{(\beta^2+1)\cdot p\cdot r}{\beta^2\cdot p+r}$$

Hierbei gibt β die relative Wichtung des Recalls an. Üblicherweise setzt man $\beta = 1$, arbeitet also mit F_1-Maß [32].
In DataMinig findet man oft folgende Formal für das F-Maß:

$$F = \frac{2\cdot p\cdot r}{p+r}$$

p: Precision
r: Recall

3.5.5 Beispiel(Konfusionsmatrix)

In der ersten Spalte sind die drei zu klassifizierenden Klassen (Klasse 1, Klasse 2, Klasse 3) und in der ersten Zeile die zu den jeweiligen Klassen zugehörigen Vorhersagen.

Aktuelle Klasse/Vorhersage	Klasse 1	Klasse 2	Klasse 3	*recall*
Klasse 1	10	2	5	r_1=0.59
Klasse 2	4	12	7	r_2=0.52
Klasse 3	3	1	9	r_3=0.69
precision	p_1=0.59	p_2=0.80	p_3=0.43	

accuracy

Die Anzahl der korrekt klassifizierten Instanzen ist die Summe der Instanzen in der Hauptdiagonale; Alle anderen sind falsch klassifiziert.

$$p = \frac{10+12+9}{10+2+5+4+12+7+3+1+9} = 0.58$$

precision

$$p_1 = \frac{10}{10+4+3} = 0.59,\ p_2 = \frac{12}{2+12+1} = 0.80,\ p_3 = \frac{9}{5+7+9} = 0.43$$

recall

$$r_1 = \frac{10}{10+2+5} = 0.59,\ r_2 = \frac{12}{12+4+7} = 0.52,\ r_3 = \frac{9}{3+1+9} = 0.69$$

3.5.6 ROC-Kurve

ROC-Kurve ist eine graphische Technik, die zur Auswertung von Data Mining-Verfahren verwendet wird. Sie wird verwendet in der Situationen, wo der Lerner versucht, Stichproben aus Testinstanzen auszuwählen, für die eine hohe Wahrscheinlichkeit positiver Ergebnisse besteht.
Die ROC-Kurve zeichnet die Anzahl der Positiven aus der Stichprobe an der vertikalen Achse an, ausgedrückt als Prozentwert der Gesamtzahl an Positiven, über der Anzahl der Negativen an der horizontalen Achse. Abbildung 8. zeigt ein Beispiel für eine ROC-Kurve- die treppenförmige Linie- für die Stichproben der Testdaten aus der Tabelle 1 [1].

[1] Tabelle 1 zeigt ein Beispiel für eine kleine Datenmenge mit 100 Instanzen, die mit einem Lernverfahren, das Wahrscheinlichkeiten für die vorhergesagte Klasse ausgibt (wie es Naive Bayes tut). Die Instanzen sind nach absteigender Wahrscheinlichkeit sortiert, gemäß der vorhergesagten Wahrscheinlichkeit einer *yes*-Antwort. Die erste Instanz ist die, von der das Lernverfahren annimmt, sie wäre am wahrscheinlichsten positiv, die zweite ist die nächst wahrscheinlichste usw. Das Lerverfahren hatte also für die Einträge 1 und 2 recht - sie sind positiv -, lag aber mit Eintrag 3 falsch - er stellte sich als negativ heraus.

rank	predicted probability	actual class
1	0.95	yes
2	0.93	yes
3	0.93	no
4	0.88	yes
5	0.86	yes
6	0.85	yes
7	0.82	yes
8	0.80	yes
9	0.80	no
10	0.79	yes
...	...	...
...	...	...

Tabelle 1: Daten für eine ROC-Kurve

Man kann ihr gemäß der Tabelle folgen. Vom Ursprung aus geht man zwei nach oben (zwei Positive), eins nach rechts (ein Negatives), fünf nach oben (fünf Positive), eins nach rechts (ein Negatives), nach oben usw. Jeder Punkt entspricht dem Zeichnen einer Linie an einer bestimmten Position auf der Rangliste und dem Zählen der darüber liegenden *yes*- und *no*- Werte, die dann vertikal und horizontal dargestellt werden.
Die diagonale Linie zeigt das erwartete Ergebnis für unterschiedlich groß *zufällig* gewählte Stichproben.

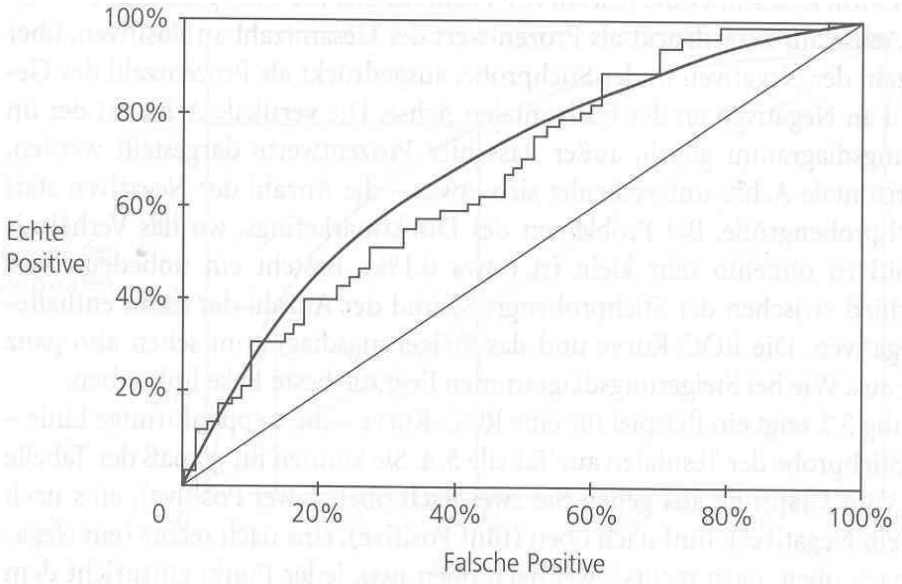


Abb.8: Beispiel für eine ROC-Kurve

Die treppenförmige ROC-Linie hängt eng mit dem Details der jeweiligen Stichproben der Datenmenge zusammen. Diese Stichprobenabhängigkeiten kann durch Anwendung der Kreuzvalidierung reduziert werden: Für jede unterschiedliche Anzahl von *no*-Werte -d.h. jede Position entlang der horizotalen Achse - man nimmt genug von der höchstrangigen Instanzen, um diese Anzahl der *no*- Werte erfüllen zu können, und zählt dann die Anzahl der enthaltenen *yes*-Werte. Schließlich mittelt man diese Zahl über unterschiedliche Durchgänge der Kreuzvalidierung. Das Ergebniss ist die glatte Kurve in Abbildung 8. ROC-Kurven demonstrieren die Leistung eines Klassifizierers ohne Berücksichtigung der Klassenverteilung oder Fehlerkosten [**?**].

3.6 Kreuzvalidierung

Für Klassifizierungsprobleme ist es sinvoll, die Leistung eines Klassifizierers hinsichtlich der Fehlerrate zu bestimmen. Der Klassifizierer sagt die Klasse jeder Instanz vorher. Liegt er richtig, wird das als Erfolg gewertet, andernfalls als Fehler. Die Fehlerrate gibt das Verhältnis an, in dem für eine Instanzmenge gemacht wurden, und misst die Leistung des Klassifizerers.
Die Fehlerrate für die Trainingdaten ist sicher keine gute Schätzung der zukünftigen Leistung, weil der Klassifizierer aus denselben Trainingdaten gelernt hat. Jede Leisungsvorhersage, die auf diesen Daten basiert ist optimistisch [69].
Um die Leistung eines Klassifizierers für neue Daten vorherzusagen, müssen wir seine Fehlerrate für eine Datenmenge ermitteln, die beim Aufbau des Klassifizierers nicht genutzt wurde. Diese unabhängige Datenmenge wird auch als *Testmenge* bezeichnet.
Die Testdaten dürfen am Erstellen des Klassifizierers nicht beteiligt sein.
Wenn sehr viel Daten zur Verfügung stehen, ist das kein Problem: Wir nehmen eine große Stichprobe und verwenden sie für das Training, dann eine weitere unabhängige große Stichprobe für den Test. Vorausgesetzt, beide Stichproben sind repräsentativ, dann stellt die Fehlerrate für die Testmenge eine zuverlässige Schätzung der zukünftigen Leistung dar.
Grob formuliert: Je größer die Trainingmenge, desto besser der Klassifizierer. Und je mehr Testdaten vorhanden sind, desto genauer ist die Schätzung des Fehlers.
Ein Problem tritt auf, wenn nicht ausreichend viele Daten zur Verfügung stehen.
Die Holdhout-Methode reserviert eine bestimmte Anzahl von Daten für das Testen und verwendet den Rest für das Training. In der Praxis ist es üblich, ein Drittel der Daten für das Testen und die restlichen zwei Drittel für das Training zu nutzen.
Man sollte sicherstellen, dass die Auswahl der Stichproben so erfolgt, dass in jedem Fall jede Klasse in der Trainings- und in der Testmenge korrekt reprä-

sentiert wird. Diese Prozedur wird auch als Stratifikation (Schichtenbildung) bezeichnet, und wir können von einem stratifizierten Holdhout sprechen.
Kreuzvalidierung ist eine statistische Technik, wobei man eine bestimmte Anzahl von Unterteilungen (Partitionen) der Daten festlegt. Die Daten werden zufällig dann z.B in drei gleich große Partitionen zerlegt, und jede Partition wird nacheinander für das Testen verwendet, während der Rest für das Training verwendet wird. Das bedeutet, man verwendet zwei Drittel für das Training und ein Drittel für das Testen und wiederholt diese Prozedur dreimal, so das letzlich jede Instanz genau einmal für das Testen verwendet wird. Man spricht dann auch von einer dreifachen Kreuzvalidierung. Wenn die Stratifikation ebenfalls angewendet wird, was häufig der Fall ist, spricht man von einer stratifizierten dreifachen Kreuzvalidierung.
Die Standardmethode, die Fehlerrate für eine Lerntechnik auf Basis einer einzigen, festen Stichprobe aus Daten vorherzusagen, ist die Anwendung einer stratifizierten zehnfachen Kreuzvalidierung. Die Daten werden durch zufällige Auswahl in zehn Teile zerlegt, wobei in jedem Teil die Klassen jeweils im annäherend selben Verhältnis repräsentiert werden wie in der vollständigen Datenmenge. Jeder dieser Teile wird dann nacheinander zurückgehalten und das Lernverfahren mit den restlichen neun Zehnteln trainiert; die Fehlerrate wird anhand der zurückgehaltenen Daten ermittelt. Die Lernprozedur wird also insgesamt zehnmal für unterschiedliche Trainingsmengen ausgeführt. Schließlich werden die zehn Fehlerschätzungen gemittelt, um die Gesamtfehlerschätzung zu erhalten.

3.7 Bezug zu anderen Themen

Statistik bzw. die statistischen Methoden kommen fast bei allen DataMining Techniken zum Einsatz wie z.B beim Bagging (Bootstrap-Aggregation). Beim Bagging (Kapitel **??**) werden nur neue Stichproben aus der ursprünglichen Datenmenge entnommen, statt aus der Domäne unabhängige Datenmengen zu erhalten. Instanzen werden aus der ursprünglichen Datenmenge zufällig -mit Ersetzen - entnommen, um eine neue Datenmenge gleicher Größe zu erzeugen [69]. Im KDD Prozess (siehe Vortrag 2) spielen die statistischen Methoden besonders in den Phasen: Datenverständnis (Verteilung der Schlüsselatribute, Beziehungen zwischen Attributen, statistische Analyse), Datenvorbereitung und vor allem in der Bewertung der verschiedenen Modelle, die in der DataMining Phase angewendet werden, eine wichtige Rolle. Die offensichtliche Methode, zwei verschiedene Lernverfahren zu vergleichen, um festzustellen, welches davon für unsere Bedürfnisse am besten geeignet ist, ist die zehnfache Kreuzvalidierung. Man wählt das Verfahren mit der kleinsten Fehlerrate aus. Da die Kreuzvalidierung nicht zuverlässig ist, verwendet man statistische Tests *t-Test,*

Chi-Quadrat-Test,...
Bei den neuartigen Methoden der Clustering (Mixture-Modelle) (Kapitel 10) wird jedes Cluster als eine Gaußsche-oder Poisson-Verteilung repräsentiert. Die Parameter dieser Verteilungen werden dabei mit den statistischen Methoden der Parameterschätzung bestimmt.
In letzter Zeit hat die Support Vektor Maschine (siehe Vortrag 5) als Lernalgorithmus an Popularität gewonnen. Dieser Algorithmus basiert auf Prinzipien der statistischen Lerntheorie und versucht obere Schranken des Generalisierungsfehlers zu minimieren. Anhand der *VC-Vapnik-Chervonenkis Dimension* lassen sich probabilistische Schranken für das Risiko angeben.
Bei der Subgruppenentdeckung (Kapitel 5), die einer der populärsten Aufgabentypen im Knowledge Discovery ist, geht es um das Finden einer Teilgruppe mit signifikant abweichendem Verhalten im Vergleich zur Gesamtpopulation bezüglich einer Zielvariable. Die Qualität von Subgruppen wird mit Hilfe statistischer Hypothesentests beurteilt. Im Lernverfahren MIDOS ist eine Hypothese die Beschreibung einer Untergruppe der Gesamtpopulation.

3.8 Zusammenfassung

Die Data Mining-Techniken, die heutzutage zum Einsatz kommen, setzen viel statistisches Denkvermögen voraus. Von Anfang an, schon wenn die Beispielmenge konstruiert und verfeinert wird, werden die Standardmethoden der Statistik angewandt: Visualiserung von Daten, Attributsauswahl, Verwerfen von Ausreißern usw. Statistische Tests werden genutzt, um Modelle und Algorithmen des maschinellen Lernens auszuwerten. Sie sind auch ein bewährtes Mittel zur explorativen Datenanalyse.
Kreuzvalidierung wird als eine statistische Technik bei der Bestimmung der Leistung für Klassifikationsverfahren eingesetzt.

Kapitel 4

Top down induction of decision trees

Siehyun Strobel

4.1 Einführung

In diesem Kapitel sollen nun als erstes Lernverfahren die sogenannten Entscheidungsbäume[1] betrachtet werden. Entscheidungsbäume finden heutzutage ein großes Anwendungsgebiet, besonders im Diagnosebereich jeglicher Art. Ein Entscheidungsbaum stellt eine einfache Datenstruktur dar, mit dem Instanzen aus einer Beispielmenge klassifiziert werden können. Dabei repräsentieren die inneren Knoten die Attribute einer Klasse und die Kanten des Baumes die entsprechenden Attributwerte.

Eine Instanz wird nun klassifiziert, indem - von der Wurzel beginnend - die Instanz 'top down' mit den Attributen bzw. den möglichen Attributwerten verglichen wird, bis schließlich ein Blatt erreicht wird. Ein Blatt enthält somit kein Attribut, sondern stellt eine Klassifikation dar, die sich auf alle Instanzen bezieht, die das Blatt erreichen.

Betrachtet man sich einen Entscheidungsbaum genauer, so sieht man leicht, dass dieser im allgemeinen eine Disjunktive Normalform (DNF) repräsentiert. Jeder Pfad von der Wurzel bis zu einem Blatt stellt eine Konjunktion der auf diesem Pfad befindlichen Attributwerte dar und der Baum selber eine Disjunktion dieser 'Klassifikationspfade'. Die folgende Abbildung eines Entscheidungsbaumes soll dies verdeutlichen.

[1] Als Grundlage für dieses Kapitel dienen die Varianten ID3 und C4.5 nach Quinlan. siehe dazu auch [52] und [69]

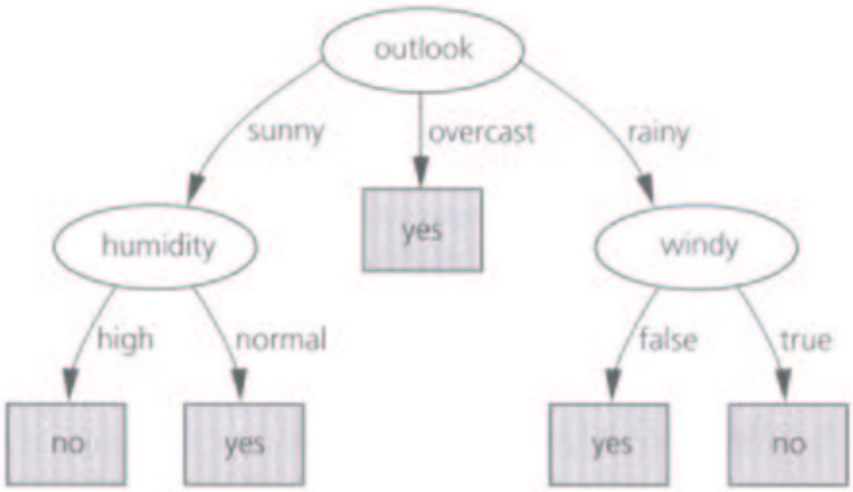


Abbildung 4.1: Beispiel Baum – siehe [69], Seite 99, Abb. 4.4

$$(\text{outlook} = \text{sunny} \wedge \text{humidity} = \text{normal}) \vee (\text{outlook} = \text{overcast}) \vee (\text{outlook} = \text{rainy} \wedge \text{windy} = \text{false})$$

Diese Eigenschaft erlaubt es also, einen Entscheidungsbaum leicht in Regeln umzuwandeln.

In einem ersten Abschnitt soll die Generierung eines solchen Entscheidungsbaumes erklärt werden. Dabei spielt der Begriff der Entropie eine zentrale Rolle, dessen Ursprung in der Informationstheorie zu finden ist. In unserem Fall wird mit der Entropie das Maß für den Informationsgehalt einer Menge S betrachtet, bzw. dessen (Un-)Reinheit. Mit diesem Maß ist es uns nun möglich, die Frage zu beantworten, welches Attribut in welchem inneren Knoten des Entscheidungsbaumes repräsentiert werden soll und stellt somit ein Auswahlkriterium dar. Nur durch eine kluge Repräsentation bleibt ein Entscheidungsbaum möglichst klein und die Arbeit mit einem solchen Baum effizient.
Auf den möglichen Wertebereich eines Attributes wird in einem anderen Abschnitt eingegangen. Ein Attribut kann von nominaler oder numerischer Natur sein. Dabei kann eine zu große Wertemenge für die Repräsentation der Attribute in den Knoten ein großes Problem darstellen, wenn z.B. ein Attribut eine Instanz eindeutig identifiziert. Man spricht in diesem Zusammenhang dann von

einem *ID-Code*. Das hat zur Folge, dass das Auswahlkriterium der Entropie dann erweitert werden muß.
Die Generierung eines Entscheidungsbaumes aus einer Trainingsmenge liefert für diese Menge immer einen optimalen Baum, der jedoch für den gesamten Suchraum eine überangepaßte Hypothese wiedergibt. Auf das Problem des *Overfitting* wird in dem letzten Abschnitt eingegangen.

4.2 Generierung von Entscheidungsbäumen

Die Frage, auf welche Art und Weise ein Entscheidungsbaum lernen soll, läßt sich auf die Problemstellung reduzieren, nach welchen Gesichtspunkten die Attribute der Trainingsmenge in diesem Baum angeordnet werden sollen. Wenn eine solche Konstruktionshilfe vorhanden ist, läßt sich ein Entscheidungsbaum leicht nach folgendem Algorithmus aufbauen:

- wähle Attribut als Wurzel
- lege für jeden möglichen Wert eine Verzweigung an
- fahre rekursiv für jede Verzweigung fort, bis alle Instanzen in einem Knoten dieselben Kriterien aufweisen

Der Algorithmus verfährt von der Wurzel an top-down und dabei rekursiv für jeden Knoten. Durch die Wahl eines Attributes wird die Instanzenmenge in Untermengen zerlegt und zwar eine für jeden möglichen Wert des Attributes. Dadurch läßt sich leicht erschließen, was ein ideales Attribut leisten soll - nämlich die Trainingsmenge auf eine Weise zu teilen, so dass die daraus entstehenden untergeordneten Knoten, d.h. dessen Kinder, möglichst rein bzgl. der Klassen sind. Dieses läßt sich mit der Entropie messen.

4.2.1 Entropie und information gain

Für die Wahl des geeigneten Attributs für einen Knoten wird ein statistisches Verfahren verwendet, das Auskunft darüber gibt, wie gut ein Attribut alleine eine Trainingsmenge klassifizert (es soll hier vorerst angenommen werden, dass es sich um nominale Attribute handelt). Eine zentrale Rolle spielt dabei der Begriff der Entropie, das für eine Menge S wie folgt definiert ist:

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

Es findet eine c-fache Klassifikation statt und $p_i$bezeichnet den Anteil von S bzgl. der Klasse i. Gemessen wird die Entropie in *bits*, die im Gegensatz zum

üblichen Gebrauch auch Bruchteile von bits annehmen können. Das ist zum Beispiel dann der Fall, wenn die Instanzen der Beispielmenge unterschiedlichen Klassen angehören. Der Entropiewert nimmt demzufolge auch nur den Wert 0 an, wenn alle Instanzen derselben Klasse zugeordnet werden. Aus diesem Grund wird die Entropie auch als Reinheitsmaß bezeichnet. Allgemein kann man sie auch als minimale Informationsmenge bezeichnen, die benötigt wird, um eine neue Instanz zu klassifizieren.
Beispiel: S sei eine Menge mit 14 Instanzen mit einer boolschen Klassifizierung. Neun Instanzen von S sind positiv und die restlichen fünf Instanzen sind negativ klassifiziert. Dann ist die Entropie von S bzgl. der boolschen Klassifikation:

$$Entropy([9,5]) = -(\frac{9}{14})log_2(\frac{9}{14}) - (\frac{5}{14})log_2(\frac{5}{14}) = 0,940$$

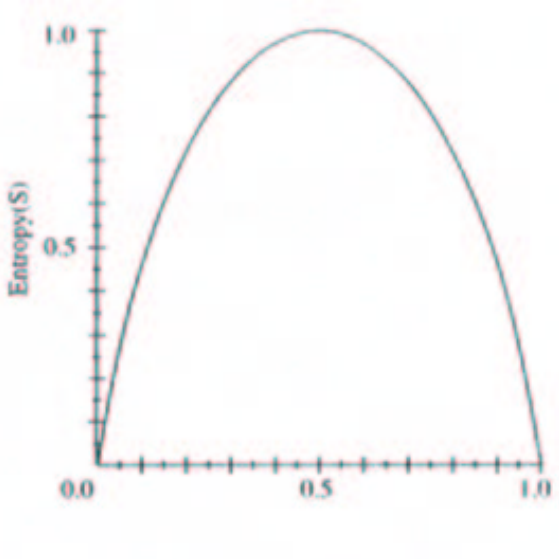


Abbildung 4.2: Entropiefunktion für c=2 – vergl. [52], Seite 57, Abb. 3.2

Mit der Entropie als Maß für die (Un-)Reinheit einer Trainingsmenge ist es nun möglich, ein weiteres Maß einzuführen, das die Effektivität eines Attributes zur Klassifikation dieser Menge misst. Man betrachtet nämlich hierzu den bisherigen Entropiewert des erstellten Entscheidungsbaumes oder bei der Wahl der Wurzel die Entropie der Trainingsmenge und vergleicht diesen mit der Entropie nach Einfügen eines Attributes. Dieser Informationsgewinn, von nun an *information gain* genannt, mißt, wie effektiv ein Attribut die Trainingsmenge separiert und damit die Entropie reduziert.
Für eine Menge S von Instanzen ist der *information gain*, *Gain (S, A)*, für ein

Attribut definiert als:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{\mid S_v \mid}{\mid S \mid} Entropy(S)$$

Values (A) ist die Menge der möglichen Werte für das Attribut A und $S_v$die Teilmenge von S, für die das Attribut den Wert v belegt ($S_v = \{s \in S \mid A(s) = v\}$). Der erste Term stellt die Entropie der gesamten Menge S dar und der zweite Term drückt die erwartete Entropie aus, nachdem das Attribut A die Menge S partitioniert hat.
Nach Vergleich der *information gain* aller Attribute, kann schließlich dasjenige 'greedy' für die Zerlegung ausgewählt werden, das den größten Informationsgewinn einbringt.
An dieser Stelle soll zum besseren Verständnis ein Beispiel betrachtet werden, nämlich das *PlayTennis*-Problem.

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Abbildung 4.3: Wettertabelle des *PlayTennis*-Problems – vgl. [69], Seite 9, Tabelle 1.2

Es soll an Hand der Attribute festgestellt werden, ob bei bestimmten Wettergegebenheiten Tennis gespielt werden soll oder nicht. Für die gesamte Trainingsmenge ist entropy ([9,5])=0,940 . Exemplarisch soll *Gain (S, outlook)* berechnet werden.

Values (outlook) = sunny, overcast, rainy

$$Entropy_{sunny}([2,3]) = 0,971bits$$

$$Entropy_{overcast}([4,0]) = 0,0bits$$

$$Entropy_{rainy}([3,2]) = 0,971bits$$

$$\begin{aligned} Gain(S, outlook) &= Entropy(S) - \sum_{v \in \{sunny, overcast, rainy\}} \frac{\mid S_v \mid}{\mid S \mid} Entropy(S_v) \\ &= Entropy(S) - (\frac{5}{14}) Entropy(S_{sunny}) - (\frac{5}{14}) Entropy(S_{rainy}) \\ &= 0,940bits - (\frac{5}{14}) * 0,971bits - (\frac{5}{14}) * 0,971bits \\ &= 0,246bits \end{aligned}$$

Analog dazu werden *Gain (S, humidity)*, *Gain (S, windy)* und *Gain (S, temperature)* berechnet:

$$\begin{aligned} Gain(S, humidity) &= 0,151bits \\ Gain(S, windy) &= 0,048bits \\ Gain(S, temperature) &= 0,029bits \end{aligned}$$

Der größten Informationsgewinn und somit die beste Vorhersage bzgl. des Zielattributs *PlayTennis* liefert das Attribut *outlook* und wird damit zum Wurzelknoten. Für jeden möglichen Wert (sunny, overcast, rainy) wird eine Verzweigung eingefügt. Der Entropiewert für das Attribut *overcast* ist Null, d.h. dass der Test mit *outlook* mit dem Wert *overcast* zu einem Blatt führt mit der Klassifizierung *PlayTennis*. Für die beiden weiteren Attributknoten muß der Entscheidungsbaum weiter betrachtet und das Auswählen des Attributs rekursiv wiederholt werden.

Abbildung 4.4: Baum mit dem Attribut *outlook* als Wurzel – vgl. [69], Seite 97, Abb. 4.2 a

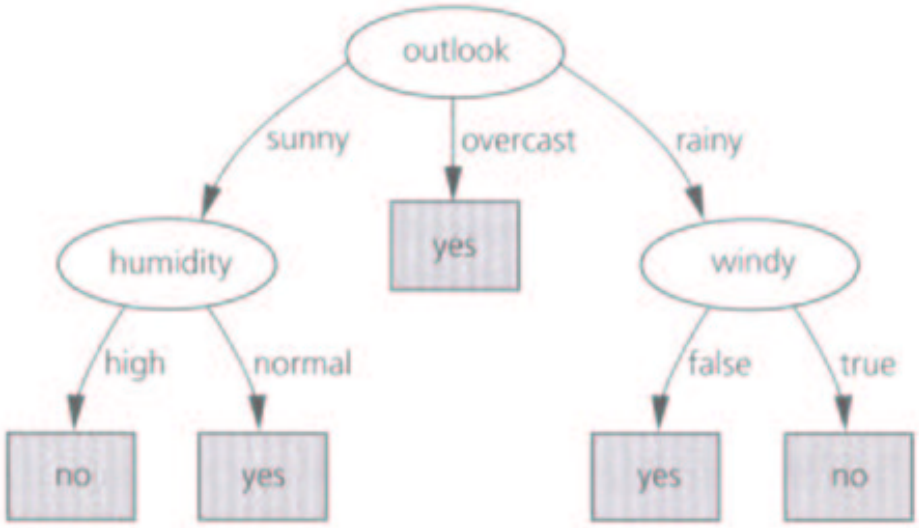


Abbildung 4.5: fertiger Entscheidungsbaum – vgl. [69], Seite 99, Abb. 4.4

4.2.2 Eigenschaften

Nach Betrachtung des Algorithmus soll nun auf die Eigenschaften eingegangen werden. Was leistet ID3 bzw. was leistet ID3 nicht?
ID3 arbeitet induktiv und durchsucht dabei einen Raum von Hypothesen nach einer, die seinen Trainingsdaten am besten entspricht. Dieser Hypothesenraum entspricht der Menge möglicher Entscheidungsbäume. Dabei sind die Entscheidungen, die von dem Algorithmus getroffen werden, immer lokal optimal, da sie durch die greedy-Eigenschaft gekennzeichnet sind. Jedoch sind diese dadurch nicht notwendigerweise auch global optimal, da einerseits nicht geprüft wird, wie viele andere - möglicherweise besser klassifizierende - alternative Hypothesen existieren und andererseits auch kein *Backtracking* vorgesehen ist. Aus diesem Grund kann keine Korrektur am gefundenen Entscheidungsbaum vorgenommen werden. Der aus einer Trainingsmenge gelernte Entscheidungsbaum kann nun dafür benutzt werden, eine beliebig andere Beispielmenge zu klassifizieren. Durch eine Generierung eines Baumes wird also die dafür genutzte Trainingsmenge generalisiert. Der Algorithmus arbeitet stets induktiv, plaziert die Attribute mit sehr hohem Informationsgewinn nah an die Wurzel und entscheidet sich für den ersten akzeptablen Baum, auf den er trifft. Das hat die Folge, dass kürzere Bäume den Längeren vorgezogen werden. Dies entspricht auch Occams These (besser bekannt als *Occam´s Razor*; zurückzuführen auf das Jahr 1320), dass kürzere Hypothesen gegenüber den Längeren favorisiert werden sollen.
Der nächste Abschnitt geht auf Ansätze ein, die die vermeintlichen Schwächen des ID-3-Algorithmus durch Erweiterungen beheben und besser als C4.5 - Algorithmus bekannt ist.

4.3 Erweiterungen des ID3-Algorithmus

4.3.1 Attribute mit großer Wertemenge

Besitzt ein Attribut eine zu große Wertemenge, kann leicht folgender Fall eintreten:
Für jede Instanz weist das Attribut einen anderen Wert auf, so dass das Attribut einen ID-Code darstellt (z.B. leicht anzutreffen bei einem Attribut, das das Datum oder ähnliches abbildet). Das Attribut identifiziert die Instanz eindeutig und der Entropiewert ergibt aus diesem Grund auch den Wert Null. Die Folge ist, dass solch ein Attribut immer den anderen vorgezogen wird, da der Informationsgewinn immer größer als die der anderen Attributen ist - nämlich genau die Information an der Wurzel.
Solch einen Extremfall gilt es natürlich zu vermeiden, da ein tatsächlicher Lerngewinn durch die Zuweisung eines solchen Knoten nicht stattfindet. Um diese

Situation zu verhindern, wird ein neues Maß eingeführt; das Gewinnverhältnis (*gain ratio*). Grundlage hierfür stellt die Splitinformation dar, die für eine Menge S und ein Attribut A wie folgt definiert ist:

$$Splitinformation(S, A) = -\sum_{i=1}^{c} \frac{\mid S_i \mid}{\mid S \mid} log_2 \frac{\mid S_i \mid}{\mid S \mid}$$

Die Splitinformation bezeichnet das Maß, auf welche Art und Weise das Attribut A die Menge S teilt. Daraus ergibt sich das Gewinnverhältnis durch

$$GainRatio(S, A) = \frac{Gain(S, A)}{Splitinformation(S, A)}$$

Das Gewinnnverhältnis berücksichtigt die Anzahl und die Größe der untergeordneten Knoten, in die ein Attribut die Menge unterteilt. Dadurch kann ein Attribut bei der Generierung eines Entscheidungsbaumes wieder ´greedy´ über die *GainRatio* ausgewählt werden.

4.3.2 Numerische Attribute

Bisher wurden nur nominale Attribte betrachtet. Die meisten Datenmengen enthalten jedoch auch numerische Attribute, wobei eine dementsprechende Erweiterung der bisher behandelten Algorithmen nicht schwer ist. Bei numerischen Attributen soll die Verzweigungsmöglichkeit auf eine binäre Aufteilung beschränkt werden, z.B. durch Vergleich mit einer Konstanten in den inneren Knoten. Dadurch stellt sich leicht die Frage, welches Attribut bzw. welche Konstante gewählt werden soll. Betrachtet man dazu ein Beispiel, so wird ersichtlich, dass man für die Wahl einer Konstanten nicht allzu viele Möglichkeiten hat.

temperature in C°	40	48	60	72	80	90
playTennis?	no	no	yes	yes	yes	no

Dieses Beispiel zeigt die Temperaturwerte der Wettertabelle - aufgetragen mit dem entsprechendem Zielattribut *playTennis*. Wünschenswert ist natürlich die Wahl einer Konstanten, die den größten *information gain* herbeiführt. Diese sind ausschließlich zwischen den Klassifikationsgrenzen zu finden; in unserem Beispiel also zwischen den Temperaturen 48 und 60 bzw. 80 und 90. Zur Wahl stehen also z.B. die Schwellwerte (48+60)/2 = 54 und (80+90)/2 = 85. Nach Berechnung der *information gain* für beide Werte, kann schließlich der Wert mit dem höheren Informationsgewinn greedy ausgewählt werden - in diesem Fall 54.

Dadurch wird bereits der erste Unterschied zwischen numerischen und nominalen Attributen ersichtlich: Die Auswertung numerischer Attribute ist nun mehrfach möglich bzw. nötig. Sie ist nicht an einer Stelle lokalisiert, sondern über den gesamten Pfad von der Wurzel eines Baumes bis zu einem Blatt verstreut, wobei bei einem nominalem Attribut bei einer Verzweigung alle Informationen ausgeschöpft wird und nur eine Auswertung über den gesamten Pfad stattfindet. Der so entstehende Baum kann deshalb leicht unübersichtlich und schwer verständlich werden. Alternativ dazu kann ein Baum gelernt werden, der Mehrfachauswertungen verschiedener Konstanten in den Knoten zuläßt. Dies führt zu einfacheren Bäumen, ist jedoch schwieriger zu realisieren.

4.3.3 Fehlende Werte

In manchen Fällen kann es vorkommen, dass in der vorhandenen Datenmenge für manche Attribute Werte fehlen. Eine Möglichkeit mit fehlenden Werten umzugehen, ist, diese einfach auf einen weiteren neuen Attributwert abzubilden, wenn das Fehlen eines Wertes von Bedeutung ist. Ist dies nicht der Fall, so ist eine andere Lösung gefragt. Ein einfacher, aber in vielen Fällen fataler Ansatz wäre alle solche Instanzen zu ignorieren. Oft liefern jedoch solche Instanzen - trotz den fehlenden Werten - noch nützliche Informationen, wenn die Attribute, deren Werte fehlen, keine Rolle bei der Entscheidung spielen. Stattdessen kann einfach der am häufigst vorkommene Wert angenommen werden. Eine andere komplexere Lösung besteht darin, eine relative Häufigkeit für jeden möglichen Attributwert zu berechnen. Diese Wahrscheinlichkeiten können durch die beobachteten Häufigkeiten der verschiedenen Werte des Attributs A am Knoten n angenommen werden. Zur Veranschaulichung soll ein boolsches Attribut A betrachtet werden. Wenn der Knoten n sechs positive Beispiele mit A=1 und vier negative mit A=0 enthält, kann die Wahrscheinlichkeit für A=1 mit 0,6 und die Wahrscheinlichkeit für A=0 mit 0,4 angegeben werden. Mit dieser entsprechenden Gewichtungen werden die Bruchteile einer Instanz auf die beiden Verzweigungen verteilt.

4.3.4 Pruning

Im allgemeinen besteht beim Lernen immer die Gefahr der Überanpassung des Lerners auf die Trainingsdaten. Bei den Entscheidungsbäumen erweitert der Lernalgorithmus die Zweige so lange bis die gesamte Trainingsmenge genau klassifiziert werden kann. Bei verrauschten Daten oder bei einer zu kleinen Trainingsmenge wird ein unzureichender und evtl. verfälschter Entscheidungsbaum erstellt. Das so entstandene Lernergebnis arbeitet zwar auf der Beispielmenge zwar optimal, kann allerdings auf der Gesamtheit der Daten schlechtere Ergebnisse liefern.

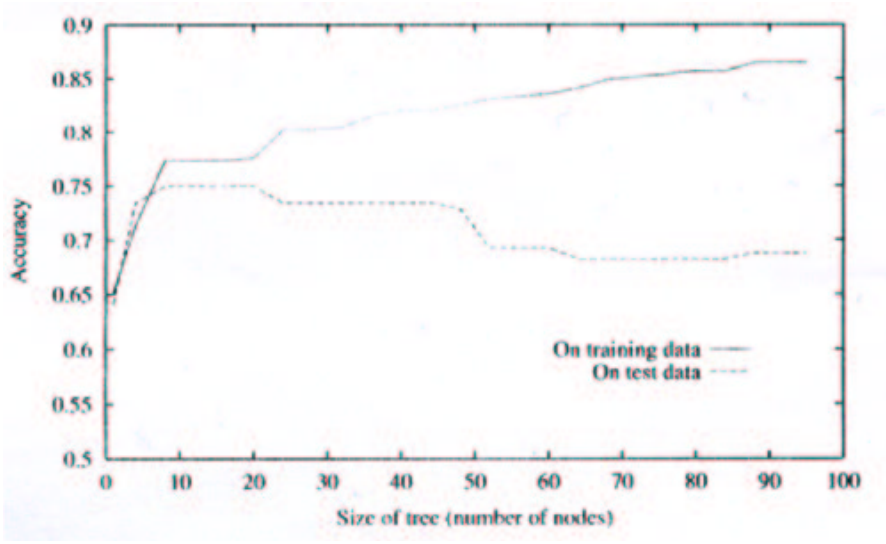


Abbildung 4.6: das *Overfitting*-Problem – vergl. [52], Seite 67, Abb.3.6

Allgemein spricht man von einer Überanpassung oder von einem *Overfitting*, wenn es zwei Entscheidungsbäume B und B´ gibt mit

- B hat auf der Trainingsmenge eine kleinere Fehlerrate als B´
- B´ hat auf dem gesamten Suchraum eine kleinere Fehlerrate als B

Der Entscheidungsbaum B ist also bzgl. den Trainingsdaten überangepaßt. Auf der nächsten Abbildung kann man das Problem des *Overfittings* beobachten. Hier wird die Größe eines Entscheidungsbaumes der *Accuracy* gegenüber gestellt. Während die *Accuracy* des Baumes über den Trainingsdaten mit der Anzahl der Knoten monoton steigt, sinkt diese immer bei einer Testdatenmenge.
Dieses Problem kann durch *Pruning* gelöst werden. Beim Pruning wird versucht, den Baum derart zu beschneiden, so dass der daraus entstandene kleinere Baum eine bessere Leistung bzgl. des Suchraums zeigt als der ursprüngliche Entscheidungsbaum.
Allgemein unterscheidet man zwischen zwei Formen des Prunings

- Pre-Pruning/Forward-Pruning
- Post-Pruning/Backward-Pruning

Beim *Pre-Pruning* oder *Forward-Pruning* wird versucht, während der Bildung des Baumes zu entscheiden, wann die Generierung von Unterbäumen eingestellt werden kann, um so einem *Overfitting* vorzubeugen. Unnötige Knoten

können somit vermieden und ein Entscheidungsbaum effizienter erstellt werden. Allerdings ist der Prozeß des Pre-Prunings sehr komplex und wird aus diesem Grund nicht oft verwendet. Aus diesem Grund wird von nun an nur das sogenannte *Post-* oder auch *Backward-Pruning* betrachtet.

Beim *Backward-Pruning* wird der gesamte Entscheidungsbaum generiert und erst dann werden unnötige Unterbäume entfernt. Hier findet eine Art Backtracking statt, das eine Korrektur von Bäumen zuläßt. Der entscheidene Vorteil gegenüber des *Forward-Prunings* ist jedoch, dass die Vorhersagekraft von Kombinationen von scheinbar ´schwachen´ Attributen erkannt wird. Es können nämlich Situationen auftreten, in denen Attribute für sich genommen eine geringe Vorhersagekraft bzgl. einer Klassifikation besitzen, allerdings zusammen in Kombination sehr aussagekräftig sind.

Für das *Backward-Pruning* stehen zwei unterschiedliche Methoden zur Verfügung: *Subtree Replacement* und *Subtree Raising*. Die Idee hinter *Subtree Replacement* ist es, Unterbäume auszuwählen und diese durch einzelne Blätter zu ersetzen. Bei einer solchen Beschneidung des Baumes sinkt natürlich die Genauigkeit auf der Trainingsmenge, da der Baum so lange von dem Generierungsalgorithmus erstellt wurde, bis dieser optimal auf der Menge arbeitet, d.h. in diesem Fall bis alle Blätter rein sind. Allerdings besteht die Hoffnung beim Subtree Replacement, dass die Accuracy bzgl. einer unabhängigen Testmenge steigt. Doch wie sieht das Verfahren des Subtree Replacements genauer aus? Dies soll an einem Beispiel, unabhängig von der Semantik, verdeutlicht werden.

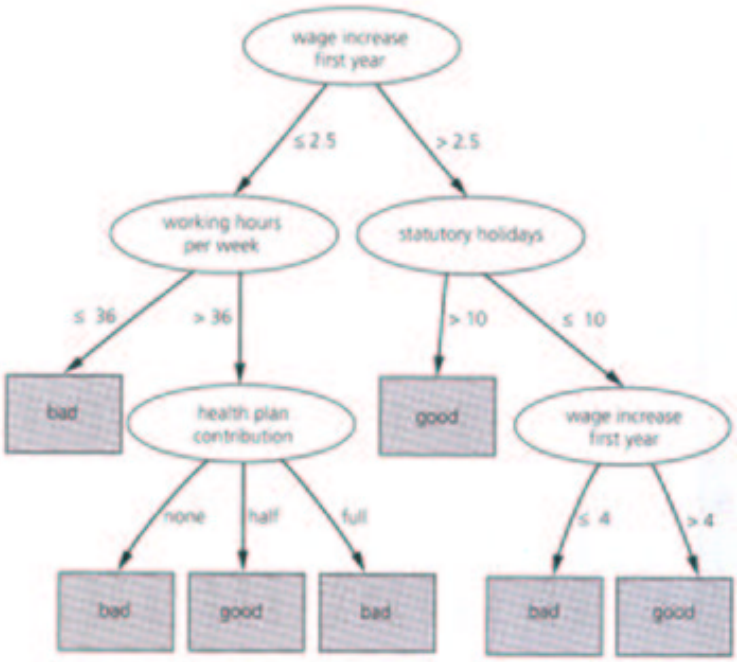


Abbildung 4.7: Entscheidungsbaum vor dem Pruning – vgl. [69], Seite 18, Abb. 1.3 a

Das *Subtree Replacement* beginnt bei den Blättern und arbeitet sich von unten zu der Wurzel hoch. Dabei wird für jeden Knoten entschieden, ob dessen Unterbaum durch ein Blatt ersetzt werden soll oder nicht. Auf einen Entscheidungsmethodik soll im nächsten Abschnitt eingegangen werden. Es wird also z.B. der Knoten *health plan contribution* betrachtet und überlegt, dessen Kinder durch ein einzelnes Blatt (z.B. *bad*) zu ersetzen. Nach Betrachtung des Knoten *working hours per week* wird nun entschieden, diesen ebenfals durch ein Blatt zu ersetzen, so dass der Baum schließlich folgende Gestalt einnimmt.

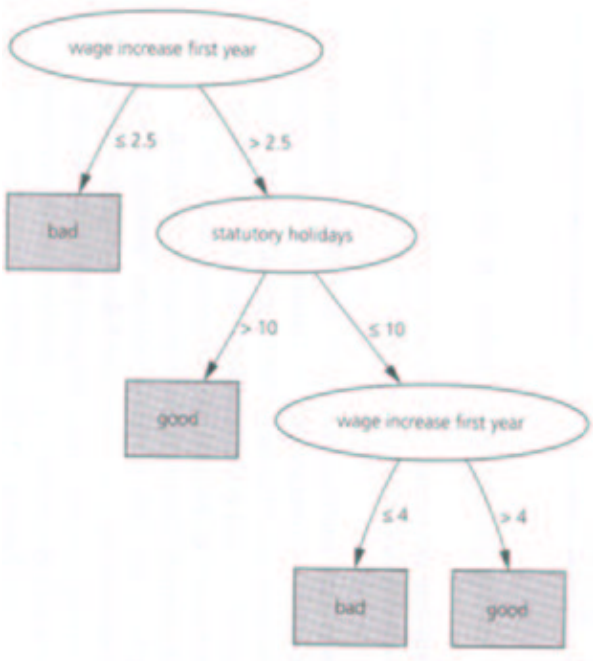


Abbildung 4.8: Entscheidungsbaum nach dem Pruning – vgl. [69], Seite 18, Abb. 1.3 b

Beim *Subtree Raising* werden Unterbäume so hochgezogen, so dass tieferliegende Knoten mit höherliegenden zusammengefaßt werden. An der folgenden Abbildung soll dies verdeutlicht werden.

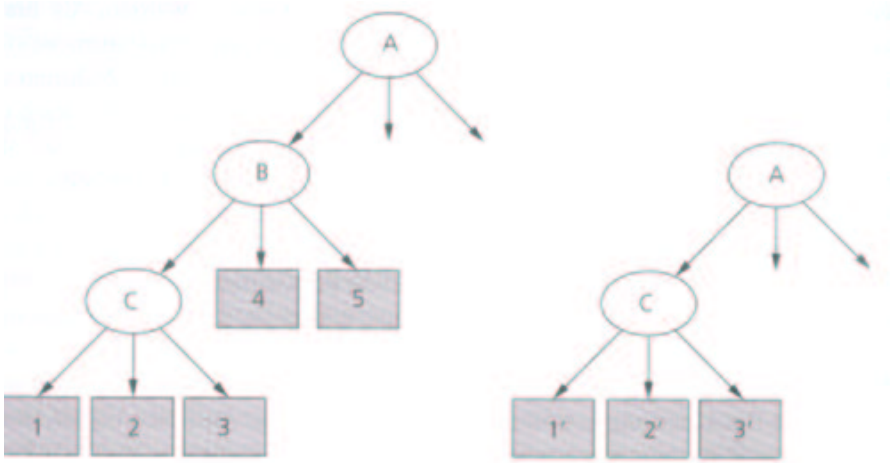


Abbildung 4.9: Beispiel für Subtree Raising – vgl. [69], Seite 176, Abb. 6.1

Der linke Baum zeigt den Baum in der ursprünglichen Form, während der rechte den Entscheidungsbaum nach einem Subtree Raising zeigt. Dabei wurde der Unterbaum unter dem Knoten *C* ´hochgezogen´ und dadurch der Unterbaum *B* ersetzt. Die neu entstandenen Blätter *1´*, *2´* und *3´* sind neu klassifiziert worden und stimmen nicht mit den ursprünglichen Knoten *1*, *2* und *3* überein, da sie die beschnittenen Blätter *4* und *5* miteinbeziehen. Subtree Raising wird in *C4.5* eingesetzt, stellt allerdings eine sehr zeitaufwendige Methodik dar, Entscheidungsbäume zu beschneiden. Aus diesem Grund werden meist nur die Unterbäume hochgezogen, dessen Verzweigungen von sehr vielen Instanzen durchlaufen werden.
Welche Frage bisher vernachlässigt wurde, ist das Problem, wie schließlich die Entscheidung getroffen wird bzw. wie weit die Bäume bei den beiden Pruning-Methoden beschnitten bzw. hochgezogen werden können. In diesem Abschnitt sollen zwei verschiedene Verfahren angesprochen werden, die der zu treffenden Entscheidung eine Fehlerrate zur Verfügung stellt, mit der abgeschätzt werden kann, ob der jeweils ursprüngliche oder der ´geprunte´ Unterbaum eine bessere Performanz erreicht. Ein Vergleich dieser Fehlerrate gibt dann Auskunft darüber, welcher Baum bzw. Unterbaum bevorzugt werden soll.

Reduced Error Pruning

Beim *Reduced Error Pruning* wird die Instanzenmenge in eine Trainings- und eine Validierungsmenge unterteilt. Mit der Trainingsmenge wird der Baum generiert, während ein Teil der Daten zurückgehalten wird, um mit dieser den Fehler an jedem Knoten abzuschätzen. Würde man diese Unterteilung nicht vornehmen und versuchen, die Fehlerrate mit der gesamten Instanzenmenge, mit der auch der Baum erstellt wurde, abzuschätzen, würde dies zu keinem

Pruning führen, weil der generierte Entscheidungsbaum auf der Menge optimal arbeitet. Stattdessen wird mit einer zurückgehaltenen Validierungsmenge gearbeitet und für jeden Knoten die Fehlerrate des Unterbaums mit der Fehlerabschätzung des möglichen ´geprunten´ Alternativ-Unterbaumes verglichen. Dabei wird der Baum so lange beschnitten, so lange die Fehlerrate sinkt.
Der Nachteil bei diesem Verfahren gründet auf der Tatsache, dass der erstellte Baum schließlich auf einer kleineren Trainingsmenge trainiert wurde und unter Umständen die zum Lernen gezogene Instanzenmenge nicht repräsentativ ist.

4.4 Bezug zu anderen Verfahren

Die Methodik der Entscheidungsbäume stellt wahrscheinlich das am gründlichsten untersuchte Lernverfahren innerhalb des Gebietes des maschinellen Lernens dar. Entscheidungsbäume können in einem Wissensentdeckungsprozeß ein erstes ´Gefühl´ für den zu untersuchenden Datensatz vermitteln - da sie zudem noch leicht zu implementieren sind, kann mit diesem Verfahren eine erste Inspektion der Daten vorgenommen werden. Außerdem kann *top down induction of decision trees* sehr gut mit anderen Lernmethoden kombiniert werden. Mit *Bagging and Boosting*(11) kann die Generierung stabiler geschehen und eine Gewichtung ohne Modifikation verarbeitet werden. Bei der *cross validation* oder der Berechnung der Entropie handelt es sich um Verfahren, dessen Ursprünge in der Statistik (3) zu finden sind.
Da Entscheidungsbäume leicht in Regeln umgewandelt werden können, können diese unter Umständen auch als Grundlage für regelbasierte Lernverfahren verwendet werden (siehe 6).
Schließlich sollte eine Merkmalsselektion bzw. Merkmalsgenerierung nicht vernachlässigt werden (siehe dazu auch 12), da diese Vorverarbeitung die Grundlage für einen erfolgreichen Wissensentdeckungsprozeß bildet. Tatsächlich können bei einer Merkmalsselektion die sogenannten ID-Attribute vor Beginn einer Entscheidungsbaumgenerierung ausgelassen werden. Zusätzlich kann ein Entscheidungsbaum für den Wrapper für die Merkmalsauswahl genutzt werden.

Kapitel 5

Support Vector Machines

Christophe Foussette

5.1 Einleitung

Support Vector Machine (deutsch: Stützvektormethode, SVM) ist ein Lernverfahren, das direkt aus der statistischen Lerntheorie hervorgegegangen ist. Diese Theorie beschreibt bezogen auf maschinelle Lernverfahren, wie gut sich ein Lernverfahren auf wahren, unbekannten Beispielen verhält, wenn es mit einer gegebenen Menge von Beispielen trainiert wurde. Die SVM kann zur Klassifikation, Regression und Dichteschätzung verwendet werden. Im folgenden wird der einfache Fall der Klassifikation betrachtet[1]. Die SVM eignet sich besonders bei Klassifikationen, deren Beispiele viele Merkmale aufweisen. Ein Beispiel ist die Bilderkennung. Die Merkmale der Beispiele können hier die Helligkeitswerte der einzelnen Pixel sein, bei einem 32x32 Pixel großen Bild ergeben sich somit 1024 Merkmale. Die Klassifikation kann darin bestehen, zu erkennen, ob auf dem Bild ein Baum vorhanden ist oder nicht. Als Trainingsbeispiele würde man Bilder verwenden, die einen oder mehrere Bäume enthalten (positive Beispiele) und Bilder die keinen Baum enthalten oder überhaupt keine Landschaftsaufnahmen darstellen (negative Beispiele). Eine weitere Anwendungsmöglichkeit der SVM ist die Texterkennung im semantischen Sinne, d.h., es sollen zum Beispiel HTML-Seiten darauf untersucht werden, ob sie sich mit der Informatik beschäftigen. In diesem Fall wären die Merkmale das Auftreten bestimmter Wörter auf der Seite. Bei einer Lernaufgabe kann nun folgendes Problem der Generalisierung auftreten, was am Beispiel der Baumerkunnung kurz erläutert werden soll. Eine gute "Auswendiglernerin" kann sich alle vorkommenden Bäume merken. D.h. sie lernt in der Lernphase alle positiven Beispiele auswendig. Wenn sie nun das Bild eines Baumes vorgelegt

[1] In Abschnitt 5.4.4 wird die Idee skizziert, um mit der SVM Regression durchzuführen.

bekommt, das sie nicht in der Lernphase auswendig gelernt hat, wird sie den Baum nicht wiedererkennen und das Bild somit falsch klassifizieren. Im Gegensatz dazu verhält sich der faule Bruder der "Auswendiglernerin". Für ihn ist alles ein Baum, was grün ist. Auch dieses Modell hat offensichtlich einen hohen Generalisierungsfehler. Es ist also für die Generalisierung von Bedeutung, welche Kapazität eine Lernmethode hat.
Der folgende Text ist im Aufbau an [11] angelehnt. Abschnitt 5.2 beschäftigt sich mit dem Generalisierungsfehler von Lernverfahren und ist somit statistisch motiviert. In Abschnitt 5.3 wird der einfache Fall von linearen SVMs erarbeitet. Abschnitt 5.4 zeigt, wie man mit Hilfe von Kernfunktionen die SVM auch auf nichtlineare Daten anwenden kann. In diesen beiden Abschnitten werden vor allem Sätze aus der Optimierungstheorie präsentiert, die die theoretische Grundlage für das SVM Training darstellen. Abschnitt 5.5 nimmt eine infomatikspezifische Sicht ein, es werden kurz Implementierungsmethoden sowie Laufzeiten angesprochen. In Abschnitt 5.6 werden für ausgewählte Kernfunktionen ihre VC Dimensionen, die in Abschnitt 5.2 eingeführt wird, berechnet. Abschnitt 5.7 verwendet die erarbeiteten Ergebnisse, um Aussagen über den Generalisierungsfehler der SVM zu treffen.

5.2 Eine Schranke für den Generalisierungsfehler

Bevor eine Schranke für den Generalisierungsfehler von maschinellen Lernverfahren und das Konzept der strukturellen Riskominimierung (SRM) vorgestellt werden, wird das Problem der Klassifikation formalisiert. Ein Lernverfahren lernt eine Entscheidungsfunktion aus bereits klassifizierten Beispielen, den Trainingsbeispielen bzw. Trainingsdaten. Ein Beispiel besteht aus einem Vektor $\mathbf{x} \in \mathbb{R}^n$, wobei n die Anzahl der Merkmale ist, und einer Klassifikation $y \in \{-1, 1\}$. -1 soll ein negatives Beispiel sein, $\mathbf{x}$ gehört nicht zur gesuchten Klasse und 1 ist ein positves Beispiel, $\mathbf{x}$ gehört zur gesuchten Klasse. Die Wahl von -1 und 1 ist mathematisch motiviert, wie sich weiter unten zeigen wird. Sie ist einfacher zu handhaben als die für einen Informatiker naheliegendere Wahl von 0 und 1. Eine Menge von l Trainingsbeispielen M ist somit eine Menge von l Paaren, bestehend aus einem Merkmalsvektor und der zugehörigen Klassifikation: $M = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$. Ein Lernverfahren soll nun die Abbildung $\mathbf{x}_i \rightarrow y_i$, $i \in 1, ..., l$, lernen. Das Lernverfahren besteht somit aus einer Menge möglicher Abbildungen $\mathbf{x} \rightarrow f(\mathbf{x}, \alpha)$, wobei α ein veränderbarer Parameter des Lernverfahrens ist. Ein festes α stellt daher ein "trainiertes Lernverfahren" dar. Bei neuronalen Netzen entspräche α den Gewichten und Schwellwerten der Neuronen. Ein Lernverfahren wird als deterministisch angenommen, wenn eine Eingabe $\mathbf{x}$ und ein gewähltes α immer das selbe Ergebnis $f(\mathbf{x}, \alpha)$ liefern.

5.2.1 Wahrer Fehler, Trainingsfehler

Der wahre Fehler, das Risiko oder der Generalisierungfehler eines Lernverfahren ist der Fehler, den das Lerverfahren auf bisher ungesehenen Beispielen macht. Das Risiko ist somit die Größe, die eine Aussage über Qualität des Lernverfahrens macht. Es wird angenommen, dass die Daten gemäß einer, in den meisten Fällen unbekannten, Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ gezogen wurden. Für das erwartete Risiko $R(\alpha)$ gilt daher:

$$R(\alpha) = \int \frac{1}{2} |y - f(\mathbf{x}, \alpha)| \, dP(\mathbf{x}, y) \tag{5.1}$$

Gleichung 5.1 ist eine einfache Darstellung des Riskos, allerdings ist sie nutzlos, wenn die Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ nicht bekannt ist, was üblicherweise der Fall ist.
Eine Größe, die ohne eine explizit angegebene Wahrscheinlichkeitsverteilung auskommt, ist das sogenannte empirische Risiko oder der Trainingsfehler $R_{emp}(\alpha)$. Dieser Trainingsfehler setzt allerdings voraus, dass sowohl Test- als auch Trainingsmenge unter der gleichen Verteilung gezogen wurden. Er ist definiert als die gemessene mittlere Fehlerrate auf den endlichen Trainingsdaten:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)| \tag{5.2}$$

Die Größe $|y_i - f(\mathbf{x}_i, \alpha)|$ wird der Verlust (loss) genannt. Im Fall der Klassifikation kann sie nur die Werte 0 und 1 annehmen. Vapnik ist es in [66] gelungen, eine obere Schranke für $R(\alpha)$ mit Hilfe von $R_{emp}(\alpha)$ zu zeigen. Hierfür wählt man ein η mit $0 \leq \eta \leq 1$. Dann gilt folgende Schranke mit Wahrscheinlichkeit $1 - \eta$:

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\left(\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l} \right)} \tag{5.3}$$

Die in Gleichung 5.3 vorkommende Größe h ist die Vapnik Chervonenkis (VC) Dimension, sie wird in Abschnitt 5.2.2 eingeführt. In der Literatur wird Gleichung 5.3 oft als "guaranteed risk" bezeichnet, was irreführend sein kann, da diese obere Schranke nur mit einer Wahrscheinlichkeit $1 - \eta$ eingehalten wird und daher nicht garantiert ist. Der zweite Summand auf der rechten Seite von 3 wird "VC confidence" genannt. Gleichung 5.3 geht zwar von einer Wahrscheinlichkeitsverteilung $P(\mathbf{x}, y)$ aus, diese muss aber nicht bekannt sein. Wenn man nun in der Lage ist, die VC Dimension h zu berechnen, kann man eine Aussage über den wahren Fehler eines Lernverfahrens machen. Formal ist ein Lernverfahren die oben vorgestellte Funktion $f(\mathbf{x}, \alpha)$. Man ist nun bestebt, die Parametereinstellung α zu finden, welche die rechte Seite von Gleichung 5.3 minimiert. Dies ist die Grundlage für die in Abschnitt 5.2.4 vorgestellte SRM.

5.2.2 VC Dimension

Die VC Dimension ist eine Größe für die Kapazität eines Lernverfahrens. Ein Lernverfahren verwendet Hypothesen aus einem Hypothesenraum, um Eingaben zu klassifizieren. So lassen sich zum Beispiel Punkte in der Ebene durch orientierte[2] Geraden aufteilen. Ein spezielle Gerade (fest gewählte Steigung und fest gewählter Achsenabschnitt) ist eine Hypothese aus dem Hypothesenraum aller Geraden. Somit kann ein Lernverfahren mit einem bestimmten Hypothesenraum identifiziert werden. Bevor nun die VC Dimension definiert wird, wird zunächst der Begriff des Zerschmetterns einer Beipielmenge eingeführt und am obigen Beispiel der Geraden erklärt.

Definition Zerschmettern einer Beispielmenge (nach [52]) Eine Menge von Beispielen S wird durch einen Hypothesenraum H (bzw. durch ein Lernverfahren bestehend aus der Menge $\{f(\alpha)\}$) genau dann *zerschmettert* (shatterd), wenn für jede Aufteilung von S in zwei Klassen eine Hypothese $h \in H$ existiert, die diese Aufteilung korrekt wiedergibt.

Was für eine Beispielmenge lässt sich nun durch orientierte Geraden in der Ebene zerschmettern? In der Definition ist die Rede von jeder Aufteilung der Beispielmenge. So lassen sich n Punkte auf 2^n verschiedene Arten aufteilen. Bei drei Punkten gibt es somit acht verschiedene Aufteilungen. In Abbildung 5.1 sind diese acht Möglichkeiten sowie die trennenden, orientierten Geraden dargestellt. An der Abbildung kann man erkennen, dass sich drei Punkte in der Ebene durch Geraden zerschmettern lassen.

Erweitert man die Beispielmenge auf vier Punkte, wie in Abbildung 5.2 gezeigt, so kann man leicht erkennen, dass die Aufteilung in die Mengen $\{x_1, x_4\}$ und $\{x_2, x_3\}$ nicht durch eine Gerade realisiert werden kann. D.h. vier Punkte in der Ebene können nicht durch Geraden zerschmettert werden.

Definition VC Dimension (nach [52] Die *VC Dimension*, $VC(H)$, eines Hypothesenraums H über einem Instanzenraum[3] X, ist die Größe der größten, endlichen Teilmenge von X, die durch H zerschmettert werden kann. Wenn eine solche größte, endliche Teilmenge nicht existiert, ist $VC(H) := \infty$.

Auf obiges Beispiel übertragen heisst dies, dass die VC Dimension von Geraden im $\mathbb{R}^2$ drei beträgt, da sich maximal drei Punkte durch Geraden zerschmettern lassen.

---

[2]Der Begriff der Orientierung resultiert hierbei aus der Klasseneinteilung der Beispiele. Geraden, oder allgemeiner Hyperebenen, können durch die Hessesche Normalform $\mathbf{w} \cdot \mathbf{x} + b = 0$ dargestellt werden. Alle Vektoren $\mathbf{x}$, die diese Gleichung erfüllen, liegen genau auf der Hyperebene. Vektoren, die diese Gleichung nicht erfüllen, sind entweder Vektoren, die die Ungleichung $\mathbf{w} \cdot \mathbf{x} + b > 0$ erfüllen, der Vektor gehört zur Klasse 1, oder Vektoren, die die Ungleichung $\mathbf{w} \cdot \mathbf{x} + b < 0$ erfüllen, der Vektor gehört zur Klasse -1. Diese Aufteilung kann invertiert werden, indem der Normalenvektor $\mathbf{w}$ mit -1 multipliziert wird.

[3]Der Instanzenraum ist der Vektorraum, dessen Elemente die Merkmalsvektoren der Beispiele sind.

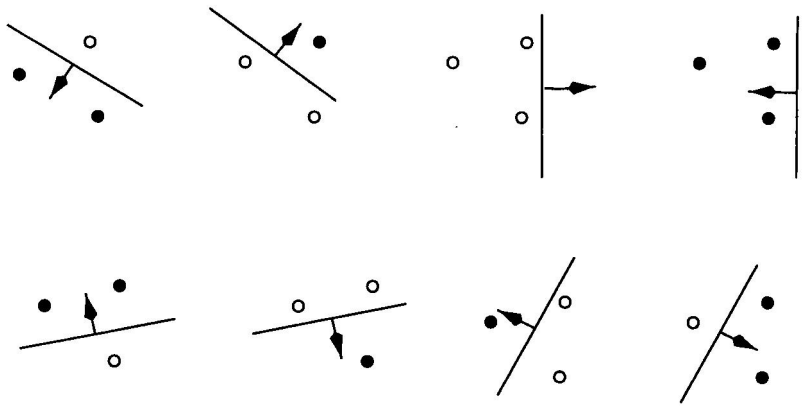


Abbildung 5.1: Drei Punkte in der Ebene mit ihren möglichen acht Aufteilungen sowie den trennenden orientierten Geraden.

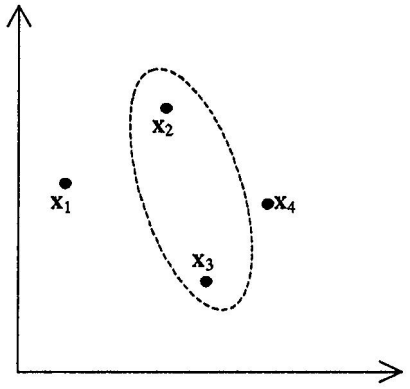


Abbildung 5.2: Vier Punkte in der Ebene, die nicht zerschmettert werden können.

VC Dimension orientierter Hyperebenen im $\mathbb{R}^n$

Das Beispiel der Geraden im $\mathbb{R}^2$ lässt sich auf orientierte Hyperebenen im $\mathbb{R}^n$ erweitern, wie der folgende Satz zeigt:

Satz 1
Sei eine Menge M von m Punkten im $\mathbb{R}^n$. Man wähle einen dieser Punkte als Urpsrung $\mathbf{u}$. Dann können die m Punkte durch eine orientierte Hyperebene zerschettert werden, genau dann, wenn die Ortsvektoren der verbleibenden Punkte $M \setminus \{\mathbf{u}\}$ linear unabhängig sind.
Für den Beweis dieses Satzes sei aus Platzgründen auf [11] verwiesen.
Korollar 1
Die VC Dimension der Menge der orientierten Hyperebenen im $\mathbb{R}^n$ ist $n + 1$,

da man immer $n+1$ Punkte, von denen einer als Ursprung gewählt wird, so wählen kann, dass die verbleibenden n Punkte linear unabhängig sind. Dagegen ist dies mit $n+2$ Punkten nicht möglich, da $n+1$ Punkte im $\mathbb{R}^n$ nicht linear unabhängig sein können.

5.2.3 Minimierung der Schranke

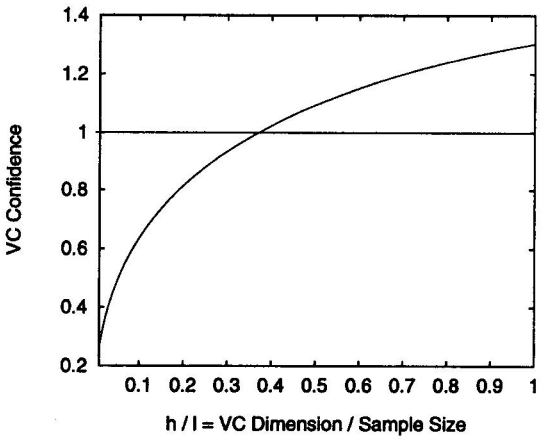


Abbildung 5.3: VC confidence aufgetragen gegen VC Dimension/Anzahl der Trainingsbeispiele

Abbildung 5.3 zeigt die VC confidence, der zweite Summand auf der rechten Seite von Gleichung 5.3, aufgetragen gegen den Quotienten VC Dimension h durch Anzahl der Trainingsdaten l (festes $l = 10000$ und $\eta = 0.05$). Man kann erkennen, dass die VC confidence eine monoton steigende Funktion in h für ein beliebiges l ist. In dem Fall, dass der Trainingsfehler einer Menge von Lernverfahren null ist, wird man natürlich bestrebt sein, das Lernverfahren aus dieser Menge zu nehmen, welches die geringste VC Dimension und somit die kleinste VC confidence besitzt. Im allgemeineren Fall, dass der Trainingsfehler größer als null ist, kommt es zu einem trade off zwischen Traingsfehler und VC confidence, da beide Werte als Summanden in die Berechnung der oberen Schranke für den wahren Fehler eingehen.
Es sei noch angemerkt, das Gleichung 5.3 nicht für jedes Lernverfahren anwendbar ist, da die Schranke für eine unendliche VC Dimension nicht gültig ist. So hat zum Beispiel der "k'th nearest neighbour" Klassifizierer zwar eine unendliche VC Dimension, hat sich in der Praxis allerdings als Lernverfahren mit guter Generalisierung bewährt [11]. Ein weiteres Beispiel für ein Lernverfahren mit unendlicher VC Dimension sind Entscheidungsbäume. Man kann zwar für stark eingeschränkte Entscheidungsbäume, zum Beispiel eingeschränkt in der Tiefe, endliche VC Dimensionen berechnen. Im allgemeinen

sind Entscheidungsbäume allerdings in der Lage, jede beliebig große Menge zu zerschmettern. Trotzdem sind Entscheidungsbäume ein gutes Lernverfahren in der Praxis. D.h. eine unendliche VC Dimension garantiert kein schlechtes Verhalten eines Lernverfahrens, im Sinne der Generalisierung.

5.2.4 Strukturelle Risikominimierung

Mit den vorangegangenen Ergebnissen kann man nun die Grundidee der SRM beschreiben. Die VC confidence hängt von der Klasse des gewählten Lernverfahrens $\{f(\mathbf{x}, \alpha)\}$ ab. Das empirische Risko hingegen hängt von der Wahl einer bestimmten Funktion, eines bestimmten α, ab. Man möchte nun die Teilmenge der gewählten Funktionen finden, deren wahrer Fehler minimal ist. Formal werden diese Teilmengen S_k durch $S_k = \{L(Y, f(\mathbf{X}, \alpha)|\alpha \in A_k\}$gebildet, wobei A_k Teilmengen der Menge aller möglichen Einstellungen für α sind, und der Struktur $S_1 \subset S_2 \subset ... \subset S_n \subset ...$ genügen. Weiterhin muss gelten, dass die VC Dimensionen h_k der einzelnen Teilmengen S_k monoton steigend sind: $h_1 \leq h_2 \leq ... \leq h_n \leq ....$ Bei der SRM wird nun die Teilmenge S_k gewählt, die den kleinsten Wert für die rechte Seite von Gleichung 5.3 liefert. Einerseits sinkt mit steigender VC Dimension das empirische Risko monoton, da mit zunehmender Kapazität des Lernverfahrens eine bessere Anpassung an die Trainingsdaten erreicht wird. Andererseits wächst mit zunehmender VC Dimension die VC confidence. Daher wird bei sukzessiver Bestimmung der rechten Seite von Gleichung 5.3 diese für jedes S_k zunächst sinken und bei einem bestimmten k_0 wieder steigen. Durch die Strukturierung der S_k wird garantiert, dass die kleinste der minimalen Riskoschranken gefunden wird.

5.3 Lineare SVMs

In diesem Abschnitt werden zunächst SVMs betrachtet, bei denen die Trainingsdaten einem linearen Modell gehorchen. Abschnitt 5.3.1 beginnt mit dem einfachsten Fall, dass sich die Merkmalsvektoren linear trennen lassen. Abschnitt 5.3.2 erweitert dies auf nicht trennbare Daten oder nach [35] ausgedrückt auf SVMs mit "weicher Trennung".

5.3.1 Trennbarer Fall

Die Trainingsdaten liegen als Paare $\{\mathbf{x}_i, y_i\}$, mit $i = 1, ..., l$, $y_i \in \{-1, 1\}$ und $\mathbf{x}_i \in \mathbb{R}^d$. Angenommen man hat eine Hyperebene, die die Trainingsdaten in positive und negative Beispiele trennt (die "trennenende Hyperebene"). Sie wird durch ihre Hessesche Normalform $\mathbf{x} \cdot \mathbf{w} + b = 0$ definiert, $\mathbf{w}$ ist der Normalenvektor und $|b| / \|\mathbf{w}\|$[4] ist ihr Abstand vom Ursprung. Seien d_+ bzw.

[4] $\|\mathbf{w}\|$ ist die euklidische Norm des Vektors $\mathbf{w}$: $\|\mathbf{w}\| := \sqrt{\mathbf{w} \cdot \mathbf{w}}$

d_- die kürzesten Abstände der nächsten positiven bzw. negativen Beispiele. Der Abstand (margin) einer trennenden Hyperebene ist definiert als $d_+ + d_-$. Im linear trennbaren Fall sucht die SVM nach der trennenden Hyperebene mit dem größten Abstand. Dies kann mit den folgenden Bedingungen ausgedrückt werden:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1, \; y_i = +1 \tag{5.4}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq -1, \; y_i = -1 \tag{5.5}$$

Die Ungleichungen 5.4 und 5.5 lassen sich zu folgender Bedingung zusammenfassen:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0, \; \forall i \tag{5.6}$$

Wenn man nun die Beispiele betrachtet, für die die Gleichheit in Ungleichung 5.4 gilt, liegen diese Beispiele auf der Hyperebene $H_1 : \mathbf{x}_i \cdot \mathbf{w} + b = 1$. Für den Abstand d_1 von H_1 zum Ursprung gilt: $d_1 = |1 - b| / \|\mathbf{w}\|$. Analog gilt für negative Beispiele, die die Gleichheit von Ungleichung 5.5 erfüllen, dass sie auf der Hyperebene $H_2 : \mathbf{x}_i \cdot \mathbf{w} + b = -1$ liegen. Somit beträgt der Abstand d_2 von H_2 zum Ursprung $| -1 - b| / \|\mathbf{w}\|$. Beide Hyperebenen sind parallel, da ihre Normalen $\mathbf{w}$ gleich sind, und keine Trainingsbeispiele befinden sich zwischen H_1 und H_2. Für den Abstand zwischen den Hyperebenen gilt nun: $d_+ + d_- = 1/\|\mathbf{w}\| + 1/\|\mathbf{w}\| = 2/\|\mathbf{w}\|$. Somit kann das Paar der Hyperebenen mit maximalem Abstand gefunden werden, indem man $\|\mathbf{w}\|^2 = \mathbf{w} \cdot \mathbf{w}$ unter Einhaltung der Bedingung 5.6 minimiert.

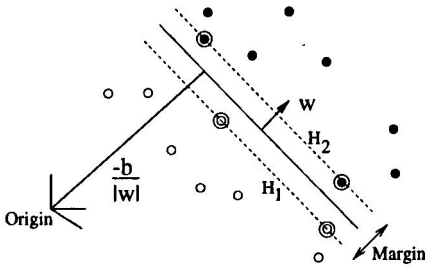


Abbildung 5.4: Linear trennende Hyperebene für den trennbaren Fall.

Dieser Sachverhalt wird in Abbildung 5.4 dargestellt. Die Vektoren, die auf den Hyperebenen H_1 und H_2 liegen, heissen Stützvektoren (support vectors, SV). In Abbildung 5.4 sind sie mit Kreisen hervorgehoben. Das besondere der SV ist, dass allein die SV die beiden Hyperebenen definieren. Selbst wenn alle anderen Beispiele weggelassen würden, würde die SVM auf die selbe trennende Hyperebene kommen.

Lagrange'sche Formulierung

Das obige Optimierungsproblem, Minimierung von $\|\mathbf{w}\|^2$ unter Berücksichtigung der Bedingung 5.6, ist mit numerischen Methoden schwer lösbar [35]. Daher wird das Optimierungsproblem mit Hilfe von Lagrange Multiplikatoren vereinfacht. Ein weiterer Grund, zu einer Lagrange'sche Formulierung zu wechseln, ist, dass die Trainingsdaten nur in Form von Skalarprodukten ihrer Merkmalsvektoren auftauchen. Dies wird sich als herausragende Eigenschaft im nichtlinearen Fall, der in Abschnitt 5.4 behandelt wird, zeigen.
Zunächst werden positive Lagrange Multiplikatoren $\alpha_i, i = 1, ..., l$ für jede Ungleichung der Bedingung 5.6 eingeführt. Man erhält folgende Lagrange Funktion:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{l} \alpha_i \tag{5.7}$$

Man muss nun L_P in Bezug auf $\mathbf{w}$ und b minimieren und gleichzeitig darauf achten, dass die Ableitungen $\frac{\partial L_P}{\partial \alpha_i}$ null werden, alles unter den Bedingungen $\alpha_i \geq 0$. Jetzt hat man ein konvexes, quadratisches Optimierungsproblem, da sowohl die Funktion selbst konvex ist, und auch die Punkte, die die Bedingungen erfüllen, eine konvexe Punktmenge bilden. Dies bedeutet, dass man das folgende, äquivalente, so genannte duale Problem lösen kann. Maximiere L_P unter der Bedingung, dass der Gradient von L_P in bezug auf $\mathbf{w}$ und b null wird, und dass $\alpha_i \geq 0$. Diese duale Formulierung des Optimierungsproblem wird Wolfe dual problem genannt [11]. Formal lässt sich die Bedingung, dass der Gradient von L_P bezüglich $\mathbf{w}$ und b null wird durch folgende zwei Gleichungen ausdrücken:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{5.8}$$

$$\sum_i \alpha_i y_i = 0 \tag{5.9}$$

Diese beiden Gleichungen können in Gleichung 5.7 eingesetzt werden, und man erhält die duale Formulierung:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{5.10}$$

SVM Training besteht nun in der Aufgabe L_D, unter den Bedingungen, dass Gleichung 5.9 gilt, und dass alle $\alpha_i \geq 0$, zu maximieren. Hierbei ist zu beachten, dass zwar für jedes Trainingsbeispiel ein Lagrange Multiplikator α_i exisiert, in der Lösung, gegeben durch Gleichung 5.8, allerdings nur die SV ein $\alpha_i > 0$ besitzen. Bei allen anderen Trainingsbeispielen sind die $\alpha_i = 0$. Dies deutet auf die zentrale Rolle der SV. Allein durch die SV wird die Entscheidungsgrenze definiert.

Karush-Kuhn-Tucker Bedingungen

Die Karush-Kuhn-Tucker (KKT) Bedingungen spielen eine zentrale Rolle in der Optimierungstheorie. Sie lauten:

$$\frac{\partial}{\partial w_k} L_P = w_k - \sum_i \alpha_i y_i x_{ik} = 0, \ k = 1, ..., d \tag{5.11}$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0 \tag{5.12}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0, \ i = 1, ..., l \tag{5.13}$$

$$\alpha_i \geq 0, \ \forall i \tag{5.14}$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0, \ \forall i \tag{5.15}$$

Bei der Lösung eines Optimierungsproblem, konvex oder nicht, sind die KKT Bedingungen erfüllt. Für konvexe Optimierungsprobleme, wie das bei der SVM auftretende, sind die KKT Bedingungen sogar notwendig und hinreichend, dass $\mathbf{w}$, b und α eine Lösung darstellen. Somit kann man, um eine Lösung für das SVM Problem zu finden, die KKT Bedingungen lösen. Der Vektor $\mathbf{w}$ wird explizit beim Training gefunden, er wird durch eine Linearkombination der SV (die Vektoren mit $\alpha_i > 0$) gebildet, Gleichung 5.8. Wie findet man nun b? Dieser Wert kann implizit aus Gleichung 5.15 berechnet werden: Man wählt ein i, für das $\alpha_i > 0$ ist, und erhält b als $b = \frac{1}{y_i} - \mathbf{x}_i \cdot \mathbf{w}$. Da Rundungsfehler auftreten können ist es sicherer, diese Prozedur für alle SV durchzuführen, und den Mittelwert der Ergebnisse als b zu verwenden.

5.3.2 Nicht trennbarer Fall

Im vorigen Abschnitt wurde angenommen, dass sich die Beispiele durch eine Hyperebene trennen lassen. Diese Annahme ist allerdings stark idealisiert, da in der Realität die Trainingsdaten zum Beispiel "verrauscht" sein können. Das oben vorgestellte Verfahren würde bei solchen Daten keine Lösung finden, da es nicht mehr konvergiert. Um auch mit realen Daten die SVM anwenden zu können, werden positive Strafterme ξ_i für jedes Beispiel $i = 1, ..., l$ eingeführt. Die Gleichungen 5.4 und 5.5 müssen somit zu folgenden Gleichungen erweitert werden:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \tag{5.16}$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \tag{5.17}$$

$$\xi_i \geq 0, \ \forall i \tag{5.18}$$

Der Strafterm ξ_i misst, wie weit der Merkmalsvektor auf der "falschen" Seite liegt. Genauer ist $\left|\frac{\xi_i}{||\mathbf{w}||}\right|$ der Abstand eines fehlerhaft klassifizierten Beispiels von

der korrekt klassifizierenden Hyperebene. Falls ein Fehler auftritt, gilt für den Starfterm $\xi_i > 1$. Daher ist die Summe $\Sigma_i \xi_i$ eine obere Schranke für die Anzahl der Trainingsfehler. Abbildung 5.5 veranschaulicht die eingeführten Größen.

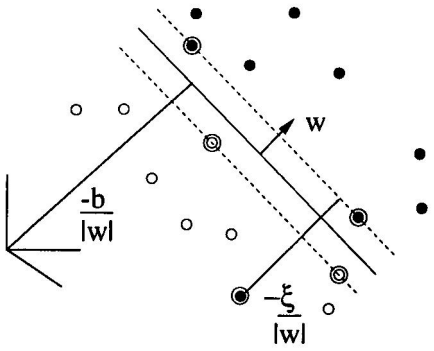


Abbildung 5.5: Linear trennende Hyperebene im nicht trennbaren Fall.

Im trennbaren Fall musste $\|\mathbf{w}\|^2/2$ minimiert werden, im nicht trennbaren Fall muss nun $\|\mathbf{w}\|^2/2 + C(\Sigma_i \xi_i)^k$ minimiert. Der hier eingeführte Wert C stellt eine Gewichtung der Fehler dar und ist ein wählbarer Parameter der SVM. Je höher C gewählt wird, umso stärker werden auftretende Fehler "bestraft". Für jede positive Wahl von k bleibt das Optimierungsproblem konvex, für $k = 1$ und $k = 2$ sogar ein quadratisches, konvexes Optimierungsproblem. Es wird $k = 1$ gewählt, da dies eine einfachere Darstellung der dualen Lagrange'schen Formulierung gewährleistet, wie man im nächsten Abschnitt sehen kann.

Lagrange'sche Formulierung

Die Wahl von $k = 1$ hat den Vorteil, dass weder die Strafterme ξ_i noch ihre Lagrange Multiplikatoren in der dualen Formulierung auftreten. Das Wolfe dual problem L_D hat nun folgende Gestalt. Maximiere

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{5.19}$$

unter den Bedingungen

$$0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0 \tag{5.20}$$

Die Lösung des Problems wird wieder durch eine Linearkombination der SV gegeben:

$$\mathbf{w} = \sum_{i=1}^{N_{SV}} \alpha_i y_i \mathbf{x}_i \tag{5.21}$$

Der größe Unterschied zum trennbaren Fall ist, dass die Lagrange Multiplikatoren α_i nun eine obere Schranke C besitzen. Um im nächsten Abschnitt die modifizierten KKT Bedingungen vorzustellen, muss man die Lagrange'sche Formulierung L_P um die Strafterme ξ_i und ihre Lagrange Multiplikatoren μ_i erweitern:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i) - \sum_i \mu_i \xi_i \quad (5.22)$$

Karush-Kuhn-Tucker Bedingungen

Die KKT Bedinungen erweitert auf den nicht trennbaren Fall lauten:

$$\frac{\partial}{\partial w_k} L_P = w_k - \sum_i \alpha_i y_i x_{ik} = 0,\ k = 1, ..., d \quad (5.23)$$

$$\frac{\partial}{\partial b} L_P = -\sum_i \alpha_i y_i = 0,\ i = 1, ..., l \quad (5.24)$$

$$\frac{\partial}{\partial \xi_i} L_P = C - \alpha_i - \mu_i = 0,\ i = 1, ..., l \quad (5.25)$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0,\ i = 1, ..., l \quad (5.26)$$

$$\xi_i \geq 0,\ \forall i \quad (5.27)$$

$$\alpha_i \geq 0,\ \forall i \quad (5.28)$$

$$\mu_i \geq 0,\ \forall i \quad (5.29)$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i) = 0,\ \forall i \quad (5.30)$$

$$\mu_i \xi_i = 0,\ \forall i \quad (5.31)$$

5.3.3 Testphase

Wie kann man eine trainierte SVM, bestimmt durch $\mathbf{w}$ und b, zur Vorhersage neuer Beispiele, Merkmalsvektoren $\mathbf{x}$, benutzen? Dazu berechnet man auf welcher Seite der Entscheidungsgrenze, sie wird gegeben durch die Hyperebene zwischen den Hyperebenen H_1 und H_2, sich der Merkmalsvektor $\mathbf{x}$ befindet. Durch die Wahl von $+1$ bzw. -1 für die positive bzw. negative Klasse, kann man dies sehr einfach mit der Signumsfunktion[5]: $y = sgn(\mathbf{w} \cdot \mathbf{x} + b)$.

[5]Die Signumsfunktion sgn ist definiert durch: $sgn(x) = 1$ für alle $x \geq 0$ und $sgn(x) = -1$ für alle $x < 0$.

5.4 Nichtlineare SVMs

Eine besondere Stärke der SVM gegenüber anderen Lernverfahren ist, dass sie auch effizient nichtlineare Zusammenhänge lernen kann. Mit effizient ist hier gemeint, dass der Rechenaufwand gegenüber dem linearen Fall nicht wesentlich steigt. Es sei nochmal darauf hingewiesen, dass die Trainingsdaten bzw. ihre Merkmalsvektoren beim im Training der SVM zu lösenden Problem, gegeben durch die Gleichungen 5.19 und 5.20, nur in Form von Skalarprodukten auftreten. Man kann die Merkmalsvektoren, angesiedelt im $\mathbb{R}^d$, nun in einen höherdimensionalen (eventuell unendlichdimensionalen) euklidischen[6] Raum H transformieren. Dies wird durch folgende Abbildung realisiert:

$$\Phi : \mathbb{R}^d \mapsto H \tag{5.32}$$

Somit kann das Training der SVM auf Skalarprodukte in H der Form $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ zurückgreifen. Dies würde den Rechenaufwand allerdings wesentlich erhöhen, falls die Dimension von H viel größer ist als d, was in der Praxis meistens der Fall ist. Diese Problematik wird mit sogenannten Kernfunktionen gelöst, die im nächsten Abschnitt vorgestellt werden.

5.4.1 Kernfunktionen

Ein Kernfunktion K ist gegeben durch $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Wenn eine solche Kernfunktion existiert, kann man in der Trainingsphase auf sie zurückgreifen, anstatt die Skalarprodukte $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ im Raum H zu berechnen. D.h. mit Hilfe der Kernfunktionen kann die SVM eine Hyperebene im Raum H berechnen, ohne die Abbildung Φ benutzen zu müssen. Darüber hinaus muss die Abbildung Φ nicht explizit bekannt sein. Zunächst ein kleines Beispiel: $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\left\|\mathbf{x_i}-\mathbf{x_j}\right\|^2/2\sigma^2}$. In diesem Fall ist der Raum H unendlich dimensional und es würde daher zu Problemen führen, explizit mit Φzu rechnen. Wenn man in den oben vorgestellten Gleichungen die Skalarprodukte $\mathbf{x}_i \cdot \mathbf{x}_j$ durch die Kernfunktion $K(\mathbf{x}_i, \mathbf{x}_j)$ ersetzt, wird der Rechenaufwand für das Training nicht wesentlich steigen. Dafür bekommt man eine SVM, die in einem unendlich dimensionalen Raum H angesiedelt ist.
Wie kann man eine auf Kernfunktionen basierende SVM zum Testen von neuen Beispielen benutzen, wo der Normalenvektor $\mathbf{w}$ der trennenden Hyperebene Element des Raumes H ist? Da beim Testen auch nur die Skalarprodukte der SV $\mathbf{s}_i$ mit dem zu testenden Merkmalsvektor $\mathbf{x}$ benötigt werden, kann man

[6] Im allgemeinen müsste man einen Hilbertraum als Bildraum benutzen. In [11] wird davon abgesehen und ein euklidischer Raum benutzt, da der Begriff Hilbertraum oft abschreckend wirkt. Ein euklidischer Raum ist ein Vektorraum mit dem aus der Schule bekannten Skalarprodukt und ist für das Verständnis von nichtlinearen SVMs ausreichend. Ein Hilbertraum ist eine Verallgemeinerung des euklidischen Raums und seine Verwendung als Bildraum wird beispielsweise in [56] benutzt.

wieder auf die Kernfunktion zurückgreifen. Somit wird die Klassifizierung eines neuen Beispiels durch das Signum der folgenden Funktion $f(\mathbf{x})$ gegeben:

$$f(\mathbf{x}) = \sum_{i=1}^{N_{SV}} \alpha_i y_i \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}_i) + b = \sum_{i=1}^{N_{SV}} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b \tag{5.33}$$

Bevor nun im nächsten Abschnitt gezeigt wird, wann eine Kernfunktion gültig ist und wann nicht, wird zunächst ein einfaches Beispiel einer Kernfunktion gezeigt, bei dem die Abbildung Φ konstruiert werden kann.
Angenommen die Daten sind Vektoren im $\mathbb{R}^2$, und man wählt die Kernfunktion $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^2$. Dann ist es leicht, einen Raum H und eine Abbildung $\Phi : \mathbb{R}^2 \mapsto H$ zu finden, die der Gleichung $(\mathbf{x}_i \cdot \mathbf{x}_j)^2 = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ genügt. Zum Beispiel ist $H = \mathbb{R}^3$ und $\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$ eine mögliche Belegung für H und Φ. Mögliche Belegung heisst, dass H und Φ für eine Kernfunktion K nicht eindeutig sind. So sind zum Beispiel auch $H = \mathbb{R}^4$ und $\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ x_1x_2 \\ x_1x_2 \\ x_2^2 \end{pmatrix}$ eine gültige Belegung für H und Φ für die gewählte Kernfunkion K.

5.4.2 Mercers Bedingung

Welche Kernfunktionen sind gültig? Oder mit anderen Worten ausgedrückt, für welche Kernfunktionen existiert ein Raum H und eine Abbidlung Φ, die den in Abschnitt 5.4.1 genannten Bedingungen genügen? Gültige Kernfunktionen sind solche, die den folgenden Satz, *Mercers Bedingung*, erfüllen:
Es existiert eine Abbildung Φ und eine Erweiterung

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \Phi(\mathbf{y})_i \tag{5.34}$$

genau dann, wenn für jede Funktion $g(\mathbf{x})$ gilt:

$$\int g(\mathbf{x})^2 d\mathbf{x} \; ist \; endlich \Rightarrow \tag{5.35}$$

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \tag{5.36}$$

Für viele Fälle ist Mercers Bedingung nicht leicht nachzuweisen, da die Implikation von 5.35 nach 5.36 für jede Funktion $g(\mathbf{x})$, die Gleichung 5.35 erfüllt, nachzuweisen ist. In [11] wird bewiesen, dass Kernfunktionen $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^p$ mit $p \geq 1$ Mercers Bedingung erfüllen. Eine Konsequenz davon ist, dass jede

Kernfunktion, die durch $K(\mathbf{x}, \mathbf{y}) = \sum_{p=0}^{\infty} c_p(\mathbf{x} \cdot \mathbf{y})^p$, wobei die c_p positive, reelle Koeffizienten sind, und die Reihe konvergiert, ausgedrückt werden kann, Mercers Bedingung erfüllt und somit eine gültige Kernfunktion darstellt.
Welche Konsequenz für die SVM hat es, wenn eine Kernfunktion nicht Mercers Bedingung erfüllt? Falls die Kernfunktion nicht Mercers Bedingung erfüllt, kann es sein, dass das quadratische Optimierungsproblem nicht konvergiert und die SVM somit zu keiner Lösung kommt.

5.4.3 Beispiele für Kernfunktionen

In gängigen Implemtierungen der SVM (zum Beispiel *mysvm*, erhältlich unter http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/) sind folgende Kernfunktionen integriert:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p \tag{5.37}$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2} \tag{5.38}$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta) \tag{5.39}$$

Gleichung 5.37 beschreibt eine *polynomielle Kernfunktion vom Grad p*, Gleichung 5.38 ist die sogenannte *gaussian radial base function* (RBF), und Gleichung 5.39 stellt eine bestimmte Art eines zweilagigen, sigmoiden neuralen Netzes dar. Im Falle der RBF Kernfunktionen ermittelt die SVM die Werte N_{SV}, $\mathbf{s}_i$, $\alpha_i$und b aus Gleichung 5.33 automatisch in der Trainingsphase. Anschaulich sind die SV $\mathbf{s}_i$ die Zentren und N_{SV} ihre Anzahl (vergleiche Abbildung). Im Falle des neuronalen Netzes besteht die erste Schicht aus N_{SV} Neuronen mit d Gewichten, wobei d die Anzahl der Merkmale ist, und die zweite Schicht besteht aus N_{SV} Gewichten, gegeben durch die α_i. Somit wird die Struktur des neuronalen Netzes in der Trainingsphase der SVM ermittelt.

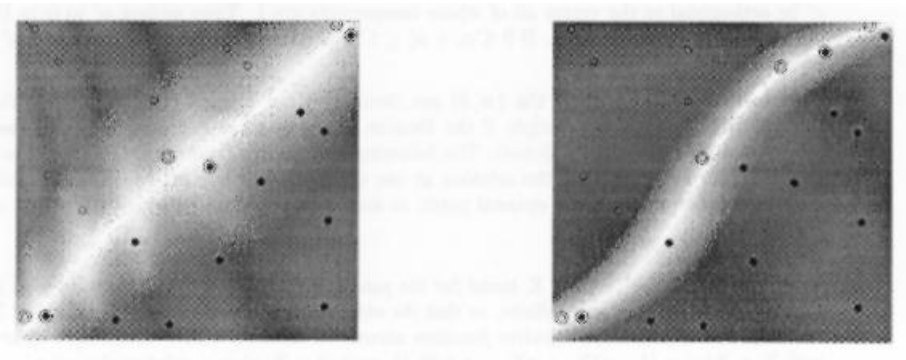


Abbildung 5.6: Trennung im nichtlinearen Fall durch eine polynomielle Kernfunktion mit Grad drei.

5.4.4 Regression mit SVMs

In diesem Abschnitt soll kurz die Idee skizziert werden, wie mit der SVM Regression betrieben werden kann. Bei der Regression liegen die l Trainingsdaten wieder in Paaren der Form $(\mathbf{x}_i, y_i)$ mit $i = 1, ..., l$ vor. Die Merkmalsvektoren $\mathbf{x}_i$ sind Elemente des $\mathbb{R}^d$. Der Unterschied zur Klassifikation besteht nun in den Werten, die die y_i annehmen können. Im Fall der Regression gilt: $y_i \in \mathbb{R}$.

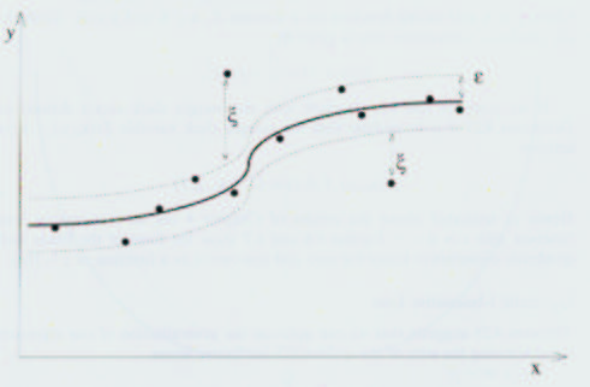


Abbildung 5.7: Eine nichtlineare Regressionsfunktion und das zugehörige ε-"Band"

In [56] wird zunächst die sogenannte ε-Insensitve Loss Regression eingeführt. Anschaulich lässt sich dies einfach anhand der Abbildung erklären. Um die Regressionsfunktion wird ein "Band" der Breite 2ε gelegt, in welchem die Trainingsbeispiele liegen dürfen. Um "Ausreisser" in den Beispielen zu berücksichtigen, werden im Fall der Regression zwei Strafterme ξ und $\hat{\xi}$ eingeführt. Basierend auf den Gleichungen 5.4 und 5.5 für den Fall der Klassifikation kann man für den Fall der Regression folgendes Optimierungsproblem herleiten:
Minimiere

$$\|\mathbf{w}\|^2 + C \sum_{i=1}^{l} (\xi_i^2 + \hat{\xi}_i^{\,2}),$$

unter den Bedingungen

$$(\mathbf{w} \cdot \mathbf{x}_i + b) - y_i \leq \varepsilon + \xi_i, \ i = 1, ..., l,$$

$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \varepsilon + \hat{\xi}_i, \ i = 1, ..., l,$$

$$\xi_i, \hat{\xi}_i \geq 0, \ i = 1, ..., l$$

Nun wird analog zu Abschnitt 5.3 folgender Weg beschritten: Das Optimierungsproblem wird durch die Lagrange'sche Formulierung in ein quadratisches, konvexes Optimierungsproblem überführt, für welches man wiederum entsprechende KKT Bedingungen herleiten kann.

5.5 Implementierung von SVMs

Es werden kurz die grundsätzlichen Schritte, die bei der Implemtierung des Trainingproblems von SVMs, zu beachten sind, skizziert. Erstens müssen die KKT Bedingungen beachtet werden, denen die Lösung (die optimale Hyperebene) genügen muss. Zweitens muss eine Strategie definiert werden, wie man sich der optimalen Lösung annähert. Hierzu kann man den Wert der dualen Lagrangefunktion unter Berücksichtigung der Nebenbedingungen schrittweise erhöhen. Schließlich sollte der Algorithmus eine Zerlegungsstrategie implementieren, so dass die benötigeten Teile der Trainingsdaten komplett im Arbeitsspeicher bearbeitet werden können.

5.5.1 Laufzeiten der Trainingsphase

[11] kommt zu folgenden Aussagen über die Laufzeiten in der Trainingsphase. Wenn die Anzahl N_{SV} der Stützvektoren gegenüber der Anzahl l der Trainingsbeispiele sehr viel kleiner ist, gilt für die Laufzeit $O(N_{SV}^3 + (N_{SV}^2)l + N_{SV}d_L l)$. Hierin ist d_L die Anzahl der Merkmale der Trainingsdaten. Wenn hingegen $N_{SV} \approx l$ gilt, beträgt die Laufzeit $O(N_{SV}^3 + N_{SV}l + N_{SV}d_L l)$.

5.5.2 Laufzeiten der Testphase

In der Testphase ist die Laufzeit durch den Rechenaufwand gegeben, der bei der Auswertung von Gleichung 5.33 auftritt. Sie beträgt $O(MN_{SV})$. Hierbei ist M die Anzahl der Rechenoperationen, die benötigt werden, um die Kernfunktion zu berechnen, M hat die Größenordnung $O(d_L)$.

5.6 Die VC Dimension von SVMs

In diesem Abschnitt wird zunächst ein allgemeiner Satz gezeigt, der die VC Dimension einer SVM abhängig von der gewählten Kernfunktion ausdrückt. Es folgen Sätze über die VC Dimension für polynomielle sowie RBF Kernfunktionen. Eine Kernfunktion kann mit einer Abbildung Φ und dem zugehörigen Bildraum H, wie in Abschnitt 5.4.1 beschrieben, assoziiert werden. Im folgenden wird unter dem *minimalen, einbettenden Raum* derjenige Raum aller möglichen Räume H verstanden, dessen Dimension d_H minimal ist. Des Weiteren sei eine gültige Kernfunktion eine, die Mercers Bedingung erfüllt und Raum L sei der Raum, aus dem die Trainingsdaten stammen.

Satz 2

Sei K eine gültige Kernfunktion mit ihrem zugehörigen minimalen, einbettenden Raum H mit Dimension d_H. Dann ist die VC Dimension der entsprechenden SVM (wobei C aus Gleichung 5.20 jeden Wert annehmen darf) $d_H + 1$.

Beweis:
Wenn der minimale einbettende Raum Dimension d_H hat, dann können d_H Punkte im Bild des Raumes $\Phi(L)$ gefunden werden, deren Ortsvektoren in H linear unabhängig sind. Nach Satz 1 können diese Vektoren durch eine Hyperebene in H zerschmettert werden. Da C jeden Wert annehmen kann, sind die α_i aus Gleichung 5.20 unbeschränkt und daher kann Satz 1 für trennende Hyperebenen angewendet werden. Demnach beträgt die VC Dimension von SVMs mit Kernel K nach Korollar 1 $d_H + 1$.

5.6.1 Die VC Dimension für polynomielle Kernfunktionen

Satz 3
Wenn der Raum L, aus dem die Merkmalsvektoren stammen, die Dimension d_L hat, gilt für die Dimension d_H des minimalen, einbettenden Raumes H für homogene polynomielle Kernfunktionen mit Grad p: $d_H = \begin{pmatrix} d_L + p - 1 \\ p \end{pmatrix}$.

Beweis: [11]
Mit Satz 2 aus Abschnitt 5.6 folgt daraus direkt:
Korollar 2
Die VC Dimension einer SVM mit homogener polynomieller Kernfunktion mit Grad p beträgt $\begin{pmatrix} d_L + p - 1 \\ p \end{pmatrix} + 1$.

5.6.2 Die VC Dimension für Radial Basis Kernfunktionen

Satz 4
Sei $\mathbf{C}$ die Klasse der gültigen Kernfunktionen mit $K(\mathbf{x}_1, \mathbf{x}_2) \to 0$ wenn $\|\mathbf{x}_1 - \mathbf{x}_2\| \to \infty$ und $K(\mathbf{x}, \mathbf{x}) = O(1)$, unter der Annahme das die Daten beliebig aus $\mathbb{R}^d$ gewählt werden. Dann haben SVMs mit Kernfunktionen der Klasse $\mathbf{C}$ (RBF Kernfunktionen gehören dieser Klasse an), wobei C aus Gleichung 5.20 jeden Wert annehmen darf, eine unendliche VC Dimension.
Beweis: [11]

5.7 Der Generalisierungsfehler von SVMs

Die VC confidence der oberen Schranke für den wahren Fehler, Gleichung 5.3 aus Abschnitt 5.2, hängt im Wesentlichen von der VC Dimension des verwendeten Lernverfahrens ab. Mit den Ergebnissen aus Abschnitt 5.6 kann somit eine obere Schranke für den wahren Fehler von SVMs, die eine polynomielle

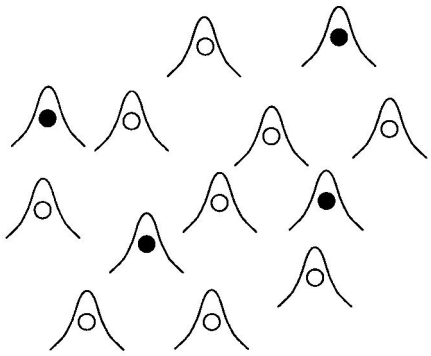


Abbildung 5.8: Gauss'sche RBF Kernfunktionen können bei genügend schmaler Breite jede beliebig große Menge von Trainigsdaten korrekt klassifizieren und haben somit eine unendliche VC Dimension.

Kernfunktion nutzen, direkt angegeben werden. Es sei nochmals darauf hingewiesen, dass diese Schranke nur mit einer wählbaren Wahrscheinlichkeit eingehalten wird. Für SVMs, die RBF Kernfunktionen benutzen, ist diese Schranke jedoch nicht gültig, da ihre VC Dimension unendlich ist. Daraus kann man allerdings nicht folgern, dass solche SVMs einen schlechten (hohen), wahren Fehler haben. Laut [11] haben sich RBF Kernfunktionen zum Beispiel bei der Bilderkennung bewährt.

5.7.1 Eine Schranke durch Leave-One-Out

Vapnik ist es in [66] gelungen eine alternative Schranke für den wahren Fehler von SVMs zu zeigen:

$$E[P(error)] \leq \frac{E[N_{SV}]}{l} \tag{5.40}$$

Hierin ist l die Anzahl der Trainingsbeispiele, $E[N_{SV}]$ ist die erwartete Anzahl der Stützvektoren über alle Trainingsmengen der Größe l und $E[P(error)]$ ist das erwartete Risiko über allen möglichen Trainingsmengen der Größe $l-1$. Wie kommt diese Schranke zustande? Bei l gegeben Beispielen kann man ein Beispiel entfernen und die SVM auf den $l-1$ verbleibenden Beispielen trainieren. Dann testet man das verbleibende Beispiel mit der so trainierten SVM. Diese Prozedur wiederholt man für alle Beispiele. Da die trennende Hyperebene bei einer SVM nur von den Stützvektoren abhängt, kann ein Testfehler nur auftreten, wenn ein Stützvektor entfernt wurde. Im worst case führt jeder als Testbeispiel zurückgehaltene Stützvektor zu einem Testfehler. Daher ist der Quotient aus der erwarteten Anzahl der Stützvektoren und der Anzahl der Trainingsdaten eine obere Schranke für den zu erwarteten Testfehler der SVM.

5.8 Schluss

Die SVM ist ein relativ neues Lernverfahren. Sie basiert zum einen auf Ergebnissen der statistischen Lerntheorie, zum anderen verwendet sie Methoden der Optimierungstheorie. Im vorliegenden Text wurde die SVM hauptsächlich zur Klassifizierung von Beispielen beschrieben. Eine Erweiterung auf den Fall der Regression wurde in Abschnitt 5.4.4 skizziert. Mit Hilfe der in Abschnitt 5.2 eingeführten VC Dimension kann der Generalisierungsfehler für spezielle SVMs nach oben beschränkt werden. Eine alternative Schranke für diesen Fehler wurde in Abschnitt 5.7.1 beschrieben. Diese theoretisch erklärbaren, statistischen Eigenschaften ermöglichen es auf SVMs die Methode der strukturellen Risikominimierung durchzuführen. Ein weiterer essentieller Vorteil der SVM ist, das mit ihr auch nichtlineare Funktionen gelernt werden können. Dies geschieht durch den Einsatz von Kernfunktionen, die in Abschnitt 5.4 vorgestellt wurden. Das Besondere an der Verwendung von Kernfunktionen ist, dass, obwohl in eventuell unendlich dimensionierten Räumen gelernt wird, die Rechenzeit der Trainings- und Testphase nicht wesentlich gegenüber dem linearen Fall steigt. Ein Nachteil der SVM ist vor allem die hohe Rechenzeit bei einer großen Anzahl von Stützvektoren (mehr als eine Million). Ein weiteres Problem besteht in der Wahl der Kernfunktion. Es ist bisher noch keine Theorie entwickelt worden, speziellen Lernaufgaben geeignete Kernfunktionen zuzuordnen. Die Wahl der Kernfunktion muss daher experimentell gerechtfertigt werden.

5.8.1 Bezug zu anderen Themen

In Kapitel 2 wird der KDD Prozess vorgestellt. In diesem Zusammenhang stellt die SVM ein mögliches Lernverfahren zum überwachten Lernen dar. D.h. die Zielgröße bei der Regression oder die Klasse bei der Klassifikation müssen bekannt sein. Ist dies nicht der Fall, müssen Lernverfahren zum unüberwachten Lernen herangezogen werden. Diese sind in den Kapiteln 6 "RDT - Rule Discovery Tool", 8 "Apriori" und 9 "Apriori für zeitliche Daten" vorgestellt. Ein weitere Einschränkung der SVM ist, dass sie nur auf numerischen Attributen lernen kann. Um auf nominalen Attributen zu lernen, muss entweder ein entsprechendes Lernverfahren herangezogen werden, oder es müssen die nominalen Attribute auf sinnvolle Weise in numerische überführt werden. Ein entsprechendes Lernverfahren ist das der im Kapitel 4 "Top down induction of decision trees" beschriebenen Entscheidungsbäume. In die Überführung von Attributen wird im Kapitel 12 "Merkmalsgenerierung" insbesondere im Abschnitt "Merkmalsgenerierung" eingeführt. Die SVM ist unter anderem durch Erkenntnisse aus der statistischen Lerntheorie, wie in Abschnitt 5.2 beschrieben, motiviert. Eine Einführung in die Begriffe der Statistik bietet das Kapitel 3 "Handwerkszeuge der Statistik". Weiterhin ist die im Abschnitt 5.7.1 vorgestellte Schranke durch Leave-One-Out eng verwandt mit der Thematik der

Kreuzvalidierung, der im zuletzt genannten Kapitel ein eigener Abschnitt gewidmet ist. Eine direkte Anwendung findet die SVM beim Clustering, das im gleichnamigen Kapitel vorgestellt wird. So wird hierbei die RBF Kernfunktion genutzt, um nach minimal einhüllenden Sphären im Bildraum des Beispielsraums zu suchen. Diese Sphären bilden nach Transformation in den Beispielraum die gesuchten Cluster.

Kapitel 6

RDT – Rule Discovery Tool

Holger Flick

6.1 Einleitung

Eine der wichtigsten praktischen Anwendungen des Maschinellen Lernens ist die interaktive Analyse von Datenbeständen geworden. Hierbei stehen vor allem die Begriffe *Data Mining* und *Knowledge Discovery in Databases* (KDD) im Vordergrund. Nach [27] wird die Wissensentdeckung in Datenbanken als ein umfassender Prozeß verstanden. Er reicht von der ersten Betrachtung der Domäne bis zur Nutzung der ermittelten Ergebnisse.
Data Mining bezeichnet die Analyse der Wissensentdeckung. Es werden Hypothesen gesucht und bewertet. Dazu werden verschiedene Verfahren des Maschinellen Lernens eingesetzt. Diese Arbeit beschreibt das Lernverfahren RDT, welches auf der induktiven logischen Programmierung basiert.
Da RDT auf der Induktiven Logischen Programmierung basiert, sind grundlegende Logik-Kenntnisse erforderlich. Somit gibt Abschnitt 6.3 einen groben Überblick über die notwendigen Kenntnisse, die mit [68] ergänzt werden sollten.
Da die Wissensentdeckung in Datenbanken in dieser Arbeit im Vordergrund stehen soll, ist auch eine Einleitung in das Gebiet der Datenbanken und die Modellierung dieser mittels logischer Strukturen unabdingbar. Hierzu ist Abschnitt 6.2 zu lesen, wobei [3, 67] zur Ergänzung dienen.
Nach der Vorstellung der Grundlagen wird dann das Verfahren RDT in Abschnitt 6.4 ausführlich vorgestellt. Die Weiterentwicklung von RDT namens RDT/DB ermöglicht die Wissensentdeckung in Datenbanken (vgl. [10]). Dieses Verfahren wird in Abschnitt 6.5 erklärt.
Im Rahmen seiner Diplomarbeit erweiterte Dirk Münstermann RDT/DB zu RDT/DM, welches in Abschnitt 6.6 kurz angerissen wird (vgl. [54]). Die-

se Arbeit beschränkt sich jedoch darauf, wesentliche Unterschiede gegenüber RDT/DB vorzustellen.
Abschließend wird im Abschnitt 6.7 versucht, die Arbeit in den Bereich der Wissensentdeckung einzuordnen und Bezüge zu anderen Gebieten in diesem Bereich herzustellen.

6.2 Datenbanken

In diesem Abschnitt sollen grundlegende Kenntnisse und Begriffe zu Datenbanken vermittelt werden. Dazu wird zunächst das *relationale Modell* vorgestellt. Anschließend werden die wichtigsten Elemente der Datenbeschreibungs- und Datenmanipulationssprache SQL aufgelistet und kurz erläutert.
Datenbanken haben sich mittlerweile in der Industrie zur Speicherung von strukturierten Datenbeständen durchgesetzt. Dies ist hauptsächlich durch das Prinzip der *Datenunabhängigkeit* und der *Datenintegration* zu begründen. (vgl. [3], [67]).
Bei der der Realisierung der Datenunabhängigkeit wird die Anwendungsentwicklung von der Datenhaltung getrennt. Dies hat den Vorteil, daß sich die Anwendungsentwickler auf das Entwickeln der Software konzentrieren können und es Spezialisten gibt, die sich nur um die optimale Datenhaltung kümmern. Meist werden in der Praxis Datenbankserver eingesetzt, die der Datenhaltung dienen und auf Anfragen der Clientrechner antworten. Elementarer Vorteil bei der Client/Server-Struktur ist, daß die verwendete Software bei den Clients vollkommen variabel gestaltet werden kann, so daß z.B. bestimmte Abteilungen einer Firma, unterschiedlichen Bedürfnissen entsprechend, unterschiedliche Software erhalten. Die Administration der Daten erfolgt jedoch zentral an einem Rechner. Dies bezeichnet man als das Prinzip der Datenintegration, d.h die Daten können aus mehreren Anwendungen genutzt werden. Eine ausführliche Erklärung und eine Abwägung der Vor- und Nachteile der beiden Prinzipen sind in [3] nachzulesen.
Aus den beiden elementaren Prinzipen kann man auch die Definition einer Datenbank ableiten (vgl. [40]):

Definition 6.2.1 (Datenbank) *Eine Datenbank ist eine Sammlung von nichtredundanten Daten, die von mehreren Applikationen genutzt werden.*

Heutzutage werden Datenbank-Management-Systeme (DBMS) eingesetzt, die die Verwaltung der Daten in der Software übernehmen. Diese Systeme enthalten dann auch eine eigene Benutzerverwaltung mit Zugriffsverwaltung.
Die Datenbank-Forschung hat sich in den letzten 20 Jahren vor allem auf die folgenden 3 Punkte konzentriert:

1. Datenmodelle und Datenbanksprache,

2. Transaktionen und Concurrency Control,

3. Datenstrukturen

Diese 3 Punkte werden meist zur Beurteilung eines DBMS herangezogen, da die verwendete Datenbanksprache sich möglichst an einem Standard orientieren sollte und die Datenstrukturen möglichst speicherplatzsparend sein und trotzdem eine gute Performance bieten sollen. Transaktionen und Parallelisierung werden von den meisten Datenbanksystemen meist automatisch, ohne direkte Aufforderung des Benutzers, eingesetzt, um höhere Geschwindigkeiten zu erziehlen. Trotzdem ist auch eine manuelle Steuerung durch den Benutzer möglich ([55]).
Für diese Arbeit ist aber vor allem das Datenmodell von Bedeutung, da Operationen mit Datenbanken formalisiert werden sollen. Dazu wird das relationale Modell verwendet.

6.2.1 Das relationale Modell

Einleitung

In dieser Arbeit soll das relationale Modell Codds vorgestellt werden. Codd wird als Urvater der relationalen Datenbanktheorie angesehen[1]. Die Grundversion des Modells veröffentlichte er in [15], erarbeitete aber in den Folgejahren zahlreiche Erweiterungen, die er in [16] und [17] veröffentlichte.
Für die Einführung des Modells beziehen wir uns auf [3] und [67], wobei die Definitionen der Begriffe aus [54] übernommen sind. Es werden zunächst die einzelnen Bestandteile vorgestellt und die für RDT Wichtigen detailliert beschrieben.

Relationen und Operatoren

Das Modell basiert auf dem mathematischen Konzept der Relation. Diese Relationen bezeichnet man dann als Tabellen, die zur Organisation der Daten eingesetzt werden. Jede Tabelle besitzt einen Namen, einen Wertebereich, die Domain, und einen Ausdehnungsgrad, den Degree. Weiterhin besteht eine Tabelle aus mehreren Spalten, die Attribute. Eine Belegung dieser Attribute mit Werten bezeichnet man als Tupel (Datensatz, Zeile). Um die Daten zu indizieren, gibt es unter Umständen einen Primärschlüssel und weitere Sekundärschlüssel. Auf die Tabellen können Operatoren angewendet werden, die zumeist aus der Mengenlehre bekannt sind: Restriktion (zur Selektion von Zeilen), Projektion (zur Selektion von Spalten), Produkt (Kartesisches Produkt), Vereinigung, Schnitt, Differenz, Verbund und Division. Mit Hilfe dieser Operationen

[1] Kurzbiographie unter URL `http://www.at-mix.de/codd_edgar.htm`

ist die relationale Algebra vollständig definiert, mit der man Operationen auf den Tabellen formalisieren kann.
Wir wollen nun eine genaue Defintion der Begriffe angeben:

Definition 6.2.2 (Tabelle) *Eine Tabelle stellt eine logisch zusammenhängende Einheit von Informationen dar. Sie besteht aus einer festen Anzahl von Attributen (Spalten) und einer variablen Anzahl von Tupeln (Zeilen). Die Relation besitzt einen eindeutigen Namen.*

Jedes Attribut besitzt einen Wertebereich, wobei einem Attribut nicht unbedingt ein Wert zugewiesen werden muß. Für diesen Fall hat man die Belegung mit 'NULL' eingeführt. Wichtig ist, daß NULL-Werte nicht verglichen werden können. Unter der Bezeichnung *Relationsschema* faßt man den Namen der Relation mit den Attributen zusammen. Die Anzahl der Attribute kennzeichnen den Ausdehnungsgrad (Degree) einer Tabelle.

Beispiel

Die folgene Tabelle hat den Namen 'players' und besitzt 3 Attribute: id, name, team. Der Wertebereich des Attributs id sind die Natürlichen Zahlen, die anderen Attribute sind beliebige Zeichenketten, die evtl. einer Längenbeschränkung unterliegen.

players

id	name	team
1	Nomo	BOS
2	Clemens	NY
3	Zito	OAK
4	Mussina	NY

Möchten wir nur die Spieler eines bestimmten Teams betrachten, so können wir über den Selektionsoperator eine Selektion durchführen:

Sel(players, team = 'NY')

id	name	team
2	Clemens	NY
4	Mussina	NY

Über eine Projektion wählen wir nur eine Spalte der Tabelle aus und können so z.B. alle Namen ermitteln:

Proj(players, < name >)

name
Nomo
Clemens
Zito
Mussina

Diese zwei vorgestellten Operationen rechtfertigen jedoch nicht wirklich die Einführung des relationalen Modells. Wünschenswert wäre es, verschiedene

Informationen auch in verschiedenen Tabellen abzulegen, jedoch trotzdem in der Lage zu sein, diese Informationen zusammenzuführen. Dies ist mit dem Join-Operator möglich.

person

id	name	gender
1	Smith, Will	m
2	Hanks, Tom	m
3	Smith, Maggie	f
4	Witherspoon, Reese	f

movies

pid	title
1	Bad Boys
3	Gosford Park
1	Enemy of the State
2	Green Mile, The
3	Harry Potter And The Phil. Stone

Die Tabelle 'person' enthält eine Liste von Schauspielern, wobei jeder Schauspieler eine eindeutige 'id' besitzt. Über diese 'id' kann in der Tabelle 'movies' nachgesehen werden, in welchem Film sie oder er mitspielt:

Join(person, movies, person.id = movies.pid)

id	name	gender	pid	title
1	Smith, Will	m	1	Bad Boys
3	Smith, Maggie	f	3	Gosford Park
1	Smith, Will	m	1	Enemy of the State
2	Hanks, Tom	m	2	Green Mile, The
3	Smith, Maggie	f	3	Harry Potter And The Phil. Stone

Man bemerkt, daß der Umfang des Ergebnisses sehr schnell groß und unübersichtlich werden kann. Dies kann in der Praxis zu starken Laufzeitbeeinträchtigungen führen, vor allem jedoch beim Maschinellen Lernen, wo meist auf großen Datenbeständen gearbeitet wird.

Es ist somit immer empfehlenswert, einen Join mit einer Projektion zu kombinieren, um nur die gewünschten Spalten in das Ergebnis zu selektieren:

Proj(Join(person, movies, person.id = movies.pid), < name, title >)

name	title
Smith, Will	Bad Boys
Smith, Maggie	Gosford Park
Smith, Will	Enemy of the State
Hanks, Tom	Green Mile, The
Smith, Maggie	Harry Potter And The Phil. Stone

Schlüssel

Der letzte Aspekt des Datenbankmodells, der wichtig im Hinblick auf die Laufzeit des Algorithmus zur Wissensentdeckung ist, sind die Schlüssel einer Tabelle.
Eine Tabelle kann Schlüssel besitzen, dies ist jedoch keine Voraussetzung. Weiterhin wird unterschieden zwischen Primär- und Sekundärschlüsseln. Grundsätzlich kann einer Tabelle nur ein Schlüssel als Primärschlüssel zugewiesen werden. Sie kann aber beliebig viele Sekundärschlüssel besitzen.

Definition 6.2.3 (Schlüssel) *Eine Menge von Attributen $\mathcal{A}$ wird als Schlüssel bezeichnet, wenn*

- *Alle Werte dieser Attributmenge sind eindeutig.*
- *$\mathcal{A}$ identifiziert jedes Tupel der Relation eindeutig.*
- *Alle Attribute der Menge $\mathcal{A}$ sind notwendig, d.h. wird ein Attribut dieser Menge entfernt, so ist die so entstandene Menge kein Schlüssel mehr.*

Schnittstelle zum Benutzer

Es wurde bisher nicht besprochen, wie der Benutzer Daten einer Datenbank einsehen oder manipulieren kann. Dafür gibt es die Datenbankanweisungen, die unterteilt werden in die Datendefinitionssprache und die Datenmanipulationssprache. Unter Datendefinitionssprache faßt man all die Befehle zusammen, die die logische Struktur bzw. Tabellen beschreiben, verändern oder löschen. Datenverarbeitende Aussagen gehören zur Datenmanipulationssprache.
Eine Vereinigung dieser beiden Sprachen wurde mit der Datenbankanfragesprache SQL realisiert. Eine ausführliche, an anschaulichen Beispielen orientierte Einführung ist in [3] angegeben. Besonders wichtig für das Thema dieser Arbeit ist der Befehl SELECT, der die Selektion, Projektion und Verknüpfung von Daten aus Tabellen ermöglicht.

6.3 Prädikatenlogik

Bevor diese Arbeit in das Thema der Wissensentdeckung einsteigen kann, sind einige grundlegende Begriffe der Prädikatenlogik abzustecken. Einen ausführlichen Überblick über die Prädikatenlogik bieten [68] und [41] (mit Beispielen und Übungen).

6.3.1 Aussagenlogik

Jede Konstante und jede Aussagenvariable ist eine atomare Formel ($A, \neg B$). Durch Operatoren können aussagenlogische Formeln miteinander verknüpft werden ($A \wedge B$). Eine Belegung der Variablen mit Werten, eine Interpretation, ermöglicht die Auswertung einer Formel. Jede Formel liefert entweder den Wert 'wahr' ($\top$) oder 'falsch' ($\bot$). Ein Literal ist eine atomare Aussage oder die Negation einer Aussagenvariablen. Die Disjunktion ($A \vee B$) von Literalen nennt man Klausel. Es gibt verschiedene Typen von Klauseln: Die Hornklausel enthält maximal 1 positives, d.h. nicht negiertes, Literal. Als Hornregel bezeichnet man eine Aussage, die genau 1 positives und mindestens ein negatives Literal aufweist. Eine Regel ist eine spezielle Form der Aussage, in der die Disjunktion als Implikation umgeschrieben wird ($\neg A \vee B \Leftrightarrow A \rightarrow B$). Ein positives Literal heißt Fakt. Prämisse nennt man die linke Seite einer Regel, Konklusion die rechte Seite.

$$A \wedge B \rightarrow C$$

In der obigen Regel bilden z.B. die Variablen A und B die Prämisse und C die Konklusion. Eine Auswertung der Aussage ist mittels einer Wertetabelle möglich.

6.3.2 Prädikatenlogik

Wollen wir nun die Aussage 'Jeder Student ist jünger als ein beliebiger Dozent?' logisch formalisieren, so fehlen uns in der Aussagenlogik dazu die Möglichkeiten, da wir dort immerzu über einen endlichen Bereich argumentieren und wir auch für jede Variable genau zwei Einsetzungsmöglichkeiten haben.
Die Prädikatenlogik bietet uns nun mit Hilfe von **Prädikaten** die Möglichkeit, logische Beziehungen und Abhängigkeiten auszudrücken:

$$student(paul),\ dozent(mike)$$

Das erste Prädikat zeichnet 'paul' als Student aus. Das zweite Prädikat sagt aus, daß 'mike' ein Dozent ist.
Variablen sind dabei dann Platzhalter für konkrete Werte:

$$student(x)\ :\ \text{x ist ein Student}$$

Es sollte nun aus den Beispielen klar werden, daß die Menge der Einsetzungsmöglichkeiten für x nicht mehr auf zwei beschränkt ist. Für x können alle möglichen Werte (eines bestimmten Universums) eingesetzt werden, wobei die Aussage nur dann wahr ist, falls für x auch wirklich ein Student eingesetzt wird. Bezogen auf den Prozeß der Wissensentdeckung in Datenbanken stellen z.B. die Tupel der Datenbank den Wertebereich dar.

Mittels **Quantoren** $(\forall, \exists)$ kann man die Beziehungen 'für alle' und 'es gibt mindestens ein' ausdrücken, was die Mächtigkeit der Prädikatenlogik noch einmal vor Augen führt.
Mit Hilfe der Prädikatenlogik können sehr leicht Verwandschaftsbeziehungen ausgedrückt werden. Will man z.B. formulieren, daß zwei Personen a und b die gleiche Großmutter haben, so kann man das nur sehr umständlich über Prädikate formalisieren. Diese Abhängigkeit läßt sich mit Funktionen einfacher ausdrücken. Eine **Funktion** liefert immer einen Term zurück, der dann wiederum mittels Funktionen ausgewertet oder von einem Prädikat umschlossen werden kann:

$$= (mutter(mutter(a)), mutter(mutter(b)))$$

Die Funktion *mutter* liefert immer die Mutter einer Person (wiederum nicht vollständig klassifiziert) zurück. Die Mutter der Mutter ist offensichtlich die Großmutter, so daß das zweistellige Prädikat $=$ die Großmütter der beiden Personen a und b vergleicht.
Die Begriffe **Term, Funktion** und **Formel** sind wie folgt abgegrenzt:

Definition 6.3.1 (Term) *1. eine Variable ist ein Term*

2. *eine Konstante ist ein Term*

3. *eine Funktion $f(t_1, \ldots, t_n)$ mit den Termen $t_1, \ldots, t_n$ ist ein Term*

4. *nichts Anderes ist ein Term!*

Definition 6.3.2 (Formel) *Die Menge der Formeln über $(\mathcal{F}, \mathcal{P})$, wobei $\mathcal{F}$ die Menge der Funktionen und $\mathcal{P}$ die Menge der Prädikate, unter Verwendung der Terme über $\mathcal{F}$:*

1. *Ist P ein Prädikat mit n Argumenten, $n \geq 1$, und wenn $t_1, \ldots, t_n$ Terme über $\mathcal{F}$, dann ist $P(t_1, \ldots, t_n)$ eine Formel*

2. *Ist α eine Formel, so auch $\neg\alpha$*

3. *Wenn α und β Formeln, so auch $\alpha \vee \beta, \alpha \wedge \beta$, und $\alpha \rightarrow \beta$*

4. *Wenn α eine Formel und x eine Variable, dann sind $\forall x\alpha$ und $\exists x\alpha$ Formeln.*

5. *nichts Anderes ist eine Formel!*

Um zu wissen, über was die verwendete Prädikatenlogik etwas aussagt, benötigt man eine Semantik. Diese Semantik bilden die **Strukturen**:

Definition 6.3.3 *Eine Struktur $\mathcal{M}$ des Paares $(\mathcal{F}, \mathcal{P})$ besteht aus der folgenden Menge:*

1. *eine nicht-leere Menge A, dem Universum der konkreten Werte (Grundbereich)*

2. *für jedes $f \in \mathcal{F}$ mit n Argumenten eine konkrete Funktion*

$$f^{\mathcal{M}} : A^n \to A$$

3. *für jedes $P \in \mathcal{P}$ mit n Argumenten eine Teilmenge $P^{\mathcal{M}} \subseteq A^n$*

Ein Beispiel für eine Struktur:

- $\mathcal{F} := \{+\}, \mathcal{P} := \{<\}$
- A sei die Menge der natürlichen Zahlen
- Regel z.B.: $< (+(a, b), +(a, c)) \to < (b, c)$

Die im Beispiel angeführte Regel ist offensichtlich. Es ist jedoch wieder zu beachten, daß Aussagen über einen unendlichen Bereich getroffen werden, wobei ganz klar spezifiziert wird, wie die Variablen substituiert werden können (nur aus dem Grundbereich $\mathcal{N}$).
Neben diesen Definitionen enthält [68] eine Auflistung nahezu aller in der Praxis relevanten Umformungsmöglichkeiten für Aussagen, worauf wir hier nicht weiter eingehen.
Die Belegung der Variablen einer prädikatenlogischen Aussage mit Werten aus dem Grundbereich der zu Grunde liegenden Struktur erfolgt durch die **Substitution**:

Definition 6.3.4 (Substitution von Aussagenvariablen) *Eine Substitution $\Theta = \{X_1|t_1, \ldots, X_n|t_n\}$ ist eine eindeutige Abbildung aus der Menge der Variablen in die Menge der Terme.*

Die Anwendung einer Substitution Θ auf eine Klausel C wird als $C\Theta$ geschrieben.
Zuletzt benötigt man noch den Begriff des Modells. Ein **Modell** ist eine Menge von Formeln (also auch Hypothesen), für die es eine Interpretation (eine Belegung mit Werten) gibt, so daß alle Formeln wahr sind. Ziel der Wissensentdeckung in Datenbanken ist es z.B. Regeln zu finden, die ein Modell über die eingesetzte Datenbank sind, d.h. alle gefundenen Regeln sind innerhalb der Datenbank erfüllt.

6.4 RDT

6.4.1 Lernen über ILP

Dem Prinzip der geordneten Suche mittels ILP liegt eine Menge von Hypothesen als Suchraum vor. Ziel der Suche ist es nun, so lange zu suchen, bis nur noch alle positiven Beispiele durch die Hypothesen abgedeckt werden bzw. keine negativen Beispiele mehr abgedeckt werden. Dies kann man durch schrittweises Generalisieren bzw. Spezialisieren der Hypothesen erreichen.
Dazu dient uns eine abgeschwächte Ableitungsvorschrift, die Subsumtion nach Plotkin.

Definition 6.4.1 *Eine Klausel C_1 subsumiert eine Klausel C_2 (formal: $C_1 \vdash_\Theta C_2$) genau dann, wenn eine Substitution Θ existiert, so daß $C_1\Theta \subseteq C_2$ gilt.*

Die Subsumtion dient somit der Generalisierung einer Klausel. Eine weitere Methode zur Generalisierung bietet die generalisierte Θ- Subsumtion, die in [37] definiert und beschrieben wird. Hervorzuheben ist, daß es sich bei der Subsumtion über eine korrekte Ableitungsvorschrift handelt, d.h. wenn $C_1 \vdash_\Theta C_2$ gilt, so folgt daraus, daß C_2 aus C_1 gefolgert werden kann. Der Umkehrschluß gilt nicht.

Beispiel

- $C_1 : tier(y) \rightarrow saeugetier(y)$
  $C_2 : tier(x) \wedge im\_haus(x) \rightarrow saeugetier(x)$
- C_2 ist spezieller als C_1, somit $C_1 \vdash_\Theta C_2$

Mit Hilfe der Subsumtion kann man auch eine Ordnung definieren, so daß die Hypothesen (z.B. beginnend mit der generellsten Hypothese bis zur speziellsten Hypothese) geordnet werden.

6.4.2 Begriff des Regellernens

RDT steht für Rule Discovery Tool und ermöglicht, wie der Name schon sagt, die Wissensentdeckung mittels Regellernen auf der Basis von Fakten. Wie die Wissensentdeckung genau definiert ist, wird in einer anderen Arbeit ausführlich bearbeitet, daher hier nur die verkürzte, informelle Definiton: [27]

Definition 6.4.2 (Regellernen) *Gegeben sei eine Menge von Beobachtungen E einer Sprache $\mathcal{L}_E$, Hintergrundwissen B in einer Sprache $\mathcal{L}_B$ sowie eine Sprache $\mathcal{L}_H$ für die Hypothesen. Gesucht wird eine Menge H von Hypothesen (Regeln) in der Sprache $\mathcal{L}_H$. Dabei muß H minimal sein in allen Modellen von B und E und jede Regel aus H muß eine neue Information über*

die Beobachtungen liefern. Weiterhin müssen alle Regeln, die in B und E wahr sind, aus H gefolgert werden können. Als letztes Kriterium wird H als minimal gefordert.

6.4.3 Metawissen

Diese Arbeit orientiert sich nun an der Gliederung der Diplomarbeit von Dirk Münstermann [54]. Aus Definition 6.4.2 folgt, daß RDT syntaktisch durch die Sprache $\mathcal{L}_H$ eingeschränkt wird. Diese Beschränkung wird durch den Benutzer vorgegeben und bildet das Metawissen. Bei dieser Art des Lernens spricht man von Lernen mit *deklarativem Bias*. Dieser kann aufgesplittet werden in den syntaktischen und semantischen Bias. Die Form der Hypothesen wird durch den syntaktischen Bias bestimmt, der semantische Bias schreibt den logischen Aufbau der Hypothesen vor. In [18] wird der Begriff des deklarativem Bias als Charakteristikum der induktiven Logischen Programmierung benannt und zeigt im maschinellen Lernen einen modellbasierten Ansatz auf. Außerdem handelt es sich bei RDT um ein Lernverfahren, daß mittels einer geordneten Suche lernt.
Die Regelschemata bilden bei RDT den syntaktischen Bias. Der semantische Bias wird durch die Prädikatentopologie und die Sortenaxonimie sichtbar.

Regelschemata

Ein Regelschema hat grundsätzlich die Form einer Regel, wobei mindestens ein Prädikatensymbol durch eine Prädikatenvariable ersetzt wurde. Jedes Regelschema hat einen Kopf mit der Bezeichnung des Schemas und mit den benutzen Prädikatenvariablen bzw. Konstanten. Ein Beispiel für ein Regelschema namens *mp* wäre z.B:

$$mp(C, P_1, P_2, P_3, Q) : P_1(X, Y) \& P_2(X, C) \& P_3(Y) \rightarrow Q(Y)$$

Das obige Regelschema benutzt 4 Prädikatenvariablen (P_1, P_2, P_3, Q) und beinhaltet eine Konstante C. X, Y seien beliebige Aussagenvariablen.
Der RDT-Algorithmus muß die Regelschemata zunächst geschickt anordnen. Daher müssen wir eine Ordnung auf Regelschemata definieren können. Dafür benötigen wir den Begriff der Relationskette und der Tiefe.

Relationskette

Man definiert eine Relationskette rekursiv. Für eine Termvariable X können u.U. mehrere Relationsketten existieren.

Definition 6.4.3 (Relationskette) *Eine Relationskette $rc(X)$ ist eine Liste von Prädikatenvariablen, die die Verbindung einer Termvariablen X mit der Konklusion darstellt. Sie ist wie folgt (rekursiv) definiert:*

1. *Ist X Termvariable der Konklusion: $rc(X) = \emptyset$.*
2. *Ist X_i mit $(1 \leq i \leq n)$ Termvariable einer Prämisse P dann gilt: $rc(X_i) = \{P\} \circ rc(X_j) \Leftrightarrow X_j$ mit $(1 \leq j \leq n)$, $j \neq i$ ist durch $rc(X_j)$ mit der Konklusion verbunden.*

Tiefe

Die Tiefe einer Termvariablen gibt die minimale Anzahl von (auszuwertenden) Prädikaten zum Erreichen der Konklusion an. Diese Größe wird für die Reihenfolge benutzt, in der die Hypothesen abgearbeitet werden.

Definition 6.4.4 *Sei X eine Termvariable. Die Tiefe $\delta(X)$ ist die Länge der minimalen Relationskette:*

$$\delta(X) = min(\{\, length(rc(X)) \,|\, rc(X) \text{ Relationskette von } X \,\})$$

Nun kann man mit Hilfe der beiden Definitionen eine Ordnung definieren, die es ermöglicht, Hypothesen gemäß der Θ-Subsumtion anzuordnen:

Prämissenordnung

Definition 6.4.5 (Prämissenordnung) *Seien P und P' Prämissen eines Regelschemas. Die Prämissenordnung $\leq_{\mathcal{P}}$ ist wie folgt definiert:*

$$P \leq_{\mathcal{P}} P' \Leftrightarrow min\left(\left\{\underbrace{\delta(X)}_{Tiefe} | X \text{ in } P\right\}\right) \leq min\left(\left\{\underbrace{\delta(X)}_{Tiefe} | X \text{ in } P'\right\}\right)$$

Die Prämissenordnung sortiert alle Prämissen gemäß ihrer kleinsten Tiefe. Daraus folgt, daß durch diese Ordnung auch die Form der Hypothesen eindeutig festgelegt wird.
Nun muß noch der logische Aufbau der Hypothesen besprochen werden. Dies tun wir über die Prädikatentopologie, durch die der Hypothesenraum eingeschränkt werden kann.

Prädikatentopologie

Jedes Lernverfahren wird zu einem bestimmten Zweck eingesetzt, damit die Ergebnisse für eine spezielle Aufgabe nützlich sind. Es ist natürlich möglich, daß der Benutzer nur relevantes Hintergrundwissen zur Verfügung stellt, jedoch kann RDT durch die Prädikatentopologie den Hypothesenraum eingrenzen und dadurch relevantes Wissen von nicht relevantem Wissen trennen.
Formal bedient man sich dem mathematischen Konstrukt des Graphen, wobei die Menge der Prädikate $\mathcal{P}$ in Mengen T_i, die Topologieknoten, eingeteilt werden. Jeder Topologieknoten bekommt eine Bezeichnung; alle Topologieknoten

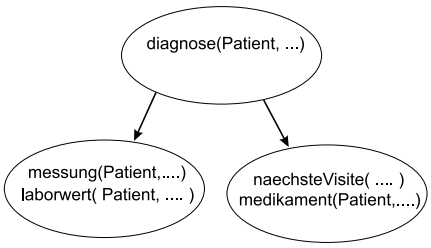


Abbildung 6.1: Beispiel für eine Wissensbasis mit 3 Topologieknoten; die beiden Blätter des Baumes decken vollkommen unterschiedliche Bereiche ab, so daß das Assoziieren von zwei Regeln aus unterschiedlichen Blättern keinen Sinn macht

werden mit einer gerichteten Kante miteinander verbunden, so daß eine Hierarchie entsteht. Grundsätzlich muß für eine Prämisse in jedem Knoten gelten, daß sie entweder in der Menge der Prämissen im eigenen Knoten oder aber in einer Menge der Vorgängerknoten enthalten ist. Formal gilt:

Definition 6.4.6 (Topologiekompatibilität) *Sei h eine Hypothese mit Konklusion q und den Prämissen p_j $(1 \leq j \leq n)$, seien $q \in T_i$, und $T_l, \ldots, T_k$ Vorgänger von T_i. Dann gilt:*

$$h \text{ ist topologiekompatibel} \Leftrightarrow \forall p_j : p_j \in T_i \vee p_j \in T_l \cup \cdots \cup T_k$$

Sortenaxonomie

Hier bedient sich RDT der Mehrsortigen Prädikatenlogik. Dazu werden Termvariablen eines Prädikates einer Sorte zugewiesen. Eine Sorte in der Logik ist vergleichbar mit einem Objekt in der Programmierung. Somit kann jedes Prädikat nur mit einer Variablen der korrekten Sorte verwendet werden, was die Anzahl der möglichen Einsetzkombinationen erheblich einschränkt.

6.4.4 Hypothesentest

Grundsätzlich wird getestet, ob eine Hypothese weiter betrachtet, also akzeptiert, oder verworfen werden kann. Eine vollinstanzierte Hypothese kann sofort auf Akzeptanz überprüft werden. Trotzdem führt RDT einen inkrementellen Test durch, bei dem Hypothesen aussortiert werden, die zu speziell sind.
Zunächst brauchen wir, wie bereits zum Ordnen der Prämissen, Maße nach denen wir überhaupt eine Bewertung vornehmen können.

Bewertungsmaße

Wir stellen Mengen zusammen, die nur Hypothesen mit einer bestimmten Eigenschaft enthalten. Wir definieren eine Hypothese $h = P(X_1, \ldots, X_m) \rightarrow q(X_1, \ldots, X_n)$, wobei $m \geq n$. Weiterhin sei $P(X_1, \ldots, X_m)$ die Konjunktion der Prämissen $p_1, \ldots, p_m$, die die Termvariablen $X_1, \ldots, X_m$ benutzen. $q(X_1, \ldots, X_n)$ ist die Konklusion mit den Termvariablen $X_1, \ldots, X_n$.

Formal	**Beschreibung**
$pos(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, q(c_1, \ldots, c_n)\}$	Menge enthält alle Belegungen, die Konklusion und Prämissen erfüllt.
$neg(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\, \neg q(c_1, \ldots, c_n)\}$	Menge enthält alle Belegungen, die Konklusion und Prämissen *nicht* erfüllt.
$pred(h) := \{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_m) \,\&\text{unknown}(q(c_1, \ldots, c_n))\}$	Menge enthält alle wahren Belegungen der Prämissen, für die die Konklusion weder als wahr noch als falsch nachweisbar ist.
$total(h) := pos(h) \cup neg(h) \cup pred(h)$	Menge aller möglichen wahren Belegungen der Prämissen
$unc(h) := \underbrace{\{(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n)\}}_{\text{Menge aller Tupel, die Konklusion erfüllen}} \setminus \underbrace{\{(c_1, \ldots, c_n) \mid P(c_1, \ldots, c_n)\}}_{\text{Menge aller Tupel, die Prämissen erfüllen}}$	Menge aller Tupel, die die Konklusion erfüllen, aber nicht von den Prämissen abgedeckt werden.
$concl(h) := \{(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n)\}$	Alle möglichen, wahren Belegungen der Konklusion

Als Bewertungsmaß wird dann die Kardinalität der jeweiligen Menge herangezogen.

Akzeptieren oder verwerfen?

Um ein Akzeptanz- oder Beschneidungskriterium anzuwenden, muß man zunächst ein wenig Erfahrung sammeln bzw. sich mit den Auswirkungen durch Anpassung der entsprechenden Werte vertraut machen. In [54] sind Beispiele für praxisrelevante Kriterien gegeben.
Es werden meist die Kardinalitäten der jeweiligen Mengen neben Operatoren auch noch mit entsprechenden Konjunktionen verbunden.
Ein äußerst relevanter Aspekt zur Bildung von Akeptanz- und Beschneidungskriterien ist die Berücksichtigung der Laufzeit, die benötigt wird, um das Kriterum auszuwerten. Legt man z.B. $|pos(h)| > 5$ als Akzeptanzkriterium fest, so kann man es mathematisch korrekt mittels $|pos(h)| \leq 5$ als Beschneidungskriterium benutzen. Leider sind die Akzeptanzkriterien nicht immer so einfach umzuformen bzw. es kann zu Situationen kommen, in denen es viel praktischer ist, bei Akzeptanz- und Beschneidungskriterium unterschiedliche Mengen als Maß einzusetzen.

6.4.5 Algorithmus

Der vollständige RDT Algorithmus ist [54] zu entnehmen. In dieser Arbeit wollen wir uns auf eine Ablaufbeschreibung beschränken.
Der RDT-Algorithmus kann in 3 grobe Abschnitte eingeteilt werden:

1. Erzeugung der Regelschemahierarchie
2. Hypothesengenerierung
3. Hypothesentest

Regelschemahierarchie

Bei der Regelschemahierarchie handelt es sich um eine Ordnung über der Menge der Regelschemata. Sie ist für den Algorithmus unabdingbar, da die Regelschemata so (umgangssprachlich gesprochen) von 'kurz' bis 'lang' bzw. 'generell' bis 'speziell' geordnet werden. Diese Hierarchie ist nur abhängig von den Regelschemata und unabhängig von der Lernaufgabe.

Definition 6.4.7 *Seien R und R' Regelschemata, σ eine Substitution bezüglich Termvariablen und Σ eine Substition bezüglich Prädikatenvariablen. Σ darf zwei verschiedene Prädikate nicht gleichsetzen.*[2] *Dann heißt R genereller als*

[2]Sei $h := \text{gelb}(X) \vee \text{blau}(x) \vee \ldots$. Ersetzt die Substitution nun 'blau' durch 'gelb', so ergibt die Aussage keinen Sinn mehr.

R' ($R \geq_{\mathcal{RS}} R'$) genau dann, wenn

$$\exists\, \sigma, \Sigma :\ R\sigma\Sigma \subseteq R'$$

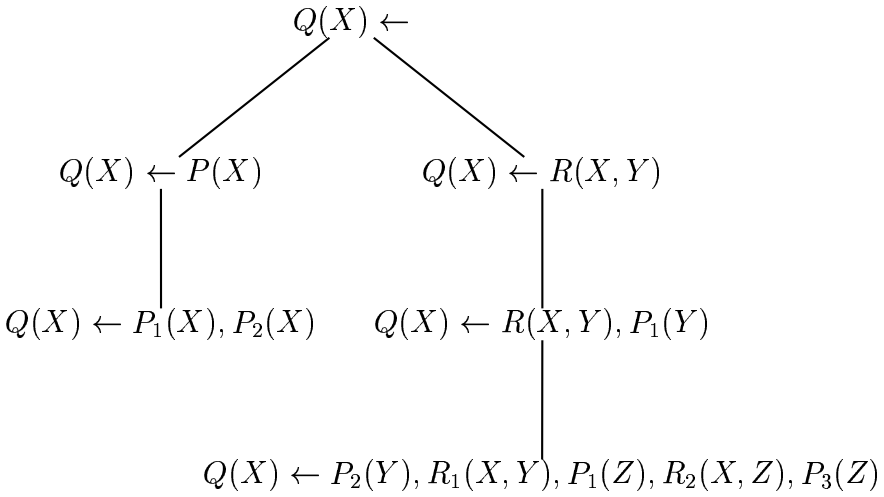


Abbildung 6.2: Beispiel für eine Regelschematahierarchie

Instanziierung

Als Instanziierung wird das Belegen der Prädikatenvariablen eines Schemas mit Prädikatensymbolen in einem Regelschema bezeichnet. Dabei wird gleichzeitig aus dem Regelschema eine Regel. Formal erfolgt die Instanziierung durch Anwendung einer Substitution:

Definition 6.4.8 *Sei P eine Prädikatenvariable, p ein Prädikatensymbol und $\mathcal{RS}$ ein Regelschema. Eine Instanziierung Σ ist eine endliche Menge von Paaren $P|p$, wobei P und p die gleiche Anzahl von Termvariablen besitzen. Mit $\mathcal{RS}\Sigma$ wird die Anwendung der Instanziierung auf das Regelschema bezeichnet. Dabei werden alle Prädikatenvariablen durch die zugeordneten Prädikatensymbole ersetzt.*

Bei Ermittlung der einsetzbaren Prädikatensymbole ist Stelligkeit, Prädikatentopoligie und Sortenkompatibilität zu berücksichtigen. Die Konstanten werden erst nach der vollständigen Instanziierung der Prädikatenvariablen belegt. Dabei werden alle möglichen konsistenten Belegungen für die Konstanten in den jeweiligen Prädikaten eingesetzt.

Testen

Es wird schrittweise instanziiert und getestet. Dies erfolgt prädikatenweise, so daß auch teil-instanziierte Regelschemata getestet werden. Die Prämissenordnung gibt dabei die Reihenfolge der zu instanziierenden Prädikate vor. Dadurch wird erreicht, daß immer das Prädikat als nächstes instanziiert wird, welches über bereits instanziierte Prädikate mit der Konklusion verbunden ist. Es werden somit keine unnötigen Instanziierungen durchgeführt, falls die verbindenen Prädikate nicht instanziiert werden können. Dazu wird die Auswahl der möglichen Instanziierungen durch die verbindenden Prädikate begrenzt. Um die Anzahl der Tests einzuschränken, werden die Hypothesen gemäß $\vdash_\Theta$ angeordnet. Erfüllt nämlich der bereits instanziierte Teil einer Hypothese den Test nicht, so müssen alle daraus resultierende Hypothesen nicht mehr berücksichtigt werden. Dies folgt unmittelbar aus der Aussage, daß es sich bei der Θ-Subsumtion um eine korrekte Ableitungsmethode handelt.

Besonderheiten

Zunächst wird dem Algorithmus ein Zielprädikat q durch den Benutzer vorgegeben. Es werden nun alle Regelschemata ausgewählt, deren Konklusion durch das Zielprädikat substituierbar ist. Anschließend wird die Regelschemahierarchie für diese Schemata aufgebaut. Diese Hierachie wird top-down mittels einer Breitensuche durchsucht. Es wird also vom generellsten zum speziellsten Regelschema untersucht. Zu beachten ist, daß jeweils nur die als zu generell eingestuften Hypothesen weiter betrachtet werden, wenn zu einem anderen Regelschema gewechselt wird. Als Datenstrukturen nutzt der Algorithumus eine triviale Liste und eine Schlange.
Die Hypothesengenerierung erfolgt gemäß dem bereits theoretisch beschriebenen Vorgehen. Auch das Testen wird ohne Modifikation der Vorüberlegungen implementiert. Der Algorithmus schaut bei einer teil-instanziierten Hypothese zunächst nach, ob es bereits eine generellere gibt, die bereits akzeptiert oder aussortiert wurde. Ist dies der Fall, so muß die Hypothese nicht weiter betrachtet werden. Andernfalls erfolgt der Test auf das Beschneidungskriterium und unter Umständen die Aussortierung. Ist die Hypothese vollständig instanziiert, so kann der Akzeptanztest erfolgen. Sollte die Hypothese akzeptiert werden, so hat der Algorithmus eine Regel gefunden. Ansonsten muß mit einem spezielleren Regelschema fortgefahren werden.

6.5 RDT/DB

Die Erweiterung von RDT zu RDT/DB wurde motiviert durch den Prozeß der Wissensentdeckung in Datenbanken. RDT/DB unterscheidet sich kaum von RDT, jedoch wird der Hypothesentest auf einer Datenbank durchgeführt.

Dazu müssen Abbildungen zwischen den Tabellen und den Prädikaten eingeführt werden. Für die Bewertungsmaße müssen außerdem Datenbankabfragen formuliert werden. [49, 10]
In dieser Arbeit sollen die Erweiterungen zu RDT herausgestellt werden. Die Definitionen wurden aus [54] übernommen.

6.5.1 Prädikatenbildung

Die Prädikate werden mittels Abbildungen gebildet. Diese sind definiert zwischen einer Tabelle in der Datenbank und den Prädikaten des Lernverfahrens. RDT/DB muß dafür zunächst die Struktur der Tabellen kennen. Kompliziertere Abbildungen beziehen auch die Attributwerte der Tabellen ein, was zum Ziel hat, daß keine Konstanten mehr gelernt werden müssen.

Abbildung Typ 1

Die erste Abbildung erzeugt für jede Tabelle genau ein Prädikat, welches die Namen der Attribute als Termvariablen benutzt.

Definition 6.5.1 *Sei R ein Tabelle mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn. Dann ist die Abbildung vom Typ 1 definiert als:*

$$R \mapsto rn(A_1, \ldots, A_n)$$

Für die Tabelle 'players' auf Seite 92 würde das Prädikat *players(id, name, team)* gebildet. Diese Abbildung ist zwar einfach zu bilden, jedoch erhalten wir auch sehr allgemeine Regeln.

Abbildung Typ 2

Um speziellere Prädikate als die vom Typ 1 zu bilden, werden bei der Abbildung vom Typ 2 die Primärschlüsselattribute berücksichtigt.

Definition 6.5.2 *Seien R eine Tabelle mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x und $A_j, \ldots, A_l$ die Primärschlüsselattribute der Tabelle. Die Abbildung vom Typ 2 ist dann definiert als:*

$$\forall x \in [1, \ldots, n] \backslash [j, \ldots, l] : R \mapsto rn\_a_x(\underbrace{A_j, \ldots, A_l}_{Primärschlüsselattr.} , \underbrace{A_x}_{verbl.\ Attribute})$$

Für jede Tabelle werden so viele Prädikate gebildet, wie es Nichtschlüsselattribute gibt. Für die Tabelle 'players' auf Seite 92 können unter der Annahme, daß das Attribut 'id' der Primärschlüssel ist, zwei Prädikate gebildet werden: *players_ name(id, name)* und *players_ team(id, team)*

Abbildung Typ 3

In Abschnitt 6.5.1 wurde von komplexeren Abbildungen gesprochen, die auch die Werte der Tabellen mit einbezieht. Die Abbildung vom Typ 3 nutzt die Werte der Tabelle, um Prädikate zu bilden. Bemerkenswert ist, daß man nun mit der Ausdrucksmächtigkeit der Aussagenlogik auskommt. Das Lernen der Konstanten, das sonst unmittelbar nach der Instanziierung der Prädikate erfolgt, ist somit nicht mehr notwendig.
Nachteil ist, daß dieser Typ der Abbildung nur für Tabellen praktikabel ist, bei denen die Anzahl der unterschiedlichen Werte in den Nichtschlüsselattributen gering ist.

Definition 6.5.3 *Seien R eine Tabelle mit den Attributen $A_1, \ldots, A_n$ und dem Namen rn, a_x der Attributname für A_x mit den Werten $a_{x1}, \ldots, a_{xm}$ und $A_j, \ldots, A_l$ die Primärschlüsselattribute der Tabelle. Dann ist die Abbildung vom Typ 3 definiert als:*

$$\forall\, x \in [1, \ldots, n] \backslash [j, \ldots, l] : \forall\, i \in [1, \ldots, m] :\ R \mapsto rn\_a_x\_a_{xi}(A_j, \ldots, A_l)$$

Für die Tabelle 'players' auf Seite 92 ist die Anwendung dieser Abbildung sinnvoll für das Attribut 'team', da es nur eine geringe Anzahl von Werten gibt. Die Spalte 'id' sei wiederum Primärschlüssel. Die Abbildung liefert nun für die Spalte 'team': *players_ team_ BOS(id), players_ team_ NY(id), players_ team_ OAK(id)*

Abbildung Typ 4

Oft sind numerische Daten in Tabellen vorzufinden. Es ist unpraktikabel, bei der Prädikatenbildung auf Werte in diesen Spalten mit der Abbildung vom Typ 3 zurückzugreifen. Aus diesem Grund bietet der RDT/DB-Algorithmus die Möglichkeit, numerische Daten in Intervalle zu unterteilen. Dies kann ausgenutzt werden, indem eine Abbildung definiert wird, die für bestimmte Intervalle einer Spalte ein Prädikat erzeugt.

6.5.2 Hypothesentest

Der Hypothesentest wird auf der Datenbank mittels SQL-Anfragen durchgeführt. Dazu wird für jedes Bewertungsmaß eine SQL-Anfrage benötigt.

Bewertungsmaße

Die Bewertungsmaße, die bei RDT/DB zum Einsatz kommen, sind nicht anders definiert als in RDT. Es werden jedoch nur noch die Kardinalitäten der Mengen *pos(h)* und *concl(h)* mit ihren Negationen *neg(h)* und *negconcl(h)* benötigt. Es ist offensichtlich, daß die Werte $|concl(h)|$ bzw. $|negconcl(h)|$ für

jede in den Regelschemata auftretende Konlusion nur einmal berechnet werden müssen. Für jede Hypothese sind immer mindestens 2 SQL-Anfragen nötig. Ist die Konklusion noch nicht instanziiert worden, sind 2 weitere Anfragen durchzuführen (vgl. Beweis in [54]).

Erzeugen der SQL-Anfragen

Es müssen nun SQL-Anfragen aus den (teil-)instanziierten Hypothesen erstellt werden. Analog zum Vorgehen zur Prädikatenbildung sind nun Abbildungen zu definieren, die eine Umformung der Prädikate in SQL-Anfragen ermöglichen. Alle verwendeten Abbildungen benötigen Informationen über das Prädikat, die assoziierte Tabelle und den zu verwendenden Abbildungstyp. Die Menge dieser Informationen sei mit $\mathcal{D}$ bezeichnet.
Es werden in RDT/DB drei Grundabbildungen definiert, die den Bezug von Prädikat(en) auf Tabelle(en) ermöglichen:

Definition 6.5.4 *1. $tab(p) : p \overset{\mathcal{D}}{\mapsto} t$; Eine Abbildung, die ein Prädikat p auf einen Tabellennamen t der zugehörigen Tabelle abbildet.*

2. $tab(P) : P \overset{\mathcal{D}}{\mapsto} T$; Eine Abbildung, die eine Prädikatenkonjunktion P auf eine Tabellenauflistung T der zugehörigen Tabellen der einzelnen Prädikate auflistet. Es wird also $tab(p)$ für jedes Element in P angewendet.

3. $pk(p) : p \overset{\mathcal{D}}{\mapsto} A$; Eine Abbildung, die ein Prädikat p auf eine Spaltennamenauflistung A abbildet, so daß die Spaltennamen den Namen der Primärschlüsselattribute der zugehörigen Tabelle entsprechen.

Zwei weitere Abbildungen werden benötigt, um den Bedingungsteil der SELECT-Anweisung erstellen zu können.

Definition 6.5.5 *1. $cond(p) : p \overset{\mathcal{D}}{\mapsto} C$; Eine Abbildung, die ein Prädikat p auf eine mit Konjunktionen verbundene Bedingungsauflistung C der zugehörigen Tabelle abbildet.*

2. $cond(P, q) : P, q \overset{\mathcal{D}}{\mapsto} C$; Eine Abbildung, die eine Prädikatenkonjunktion P und ein zusätzliches Prädikat q auf eine mit Konjunktionen verbundene Bedingungsauflistung C der zugehörigen Tabellen der einzelnen Prädikate abbildet.

Die SQL-Anweisungen können nun definiert werden:

Definition 6.5.6 *Sei $h = P(X_1, \ldots, X_m) \rightarrow q(X_1, \ldots, X_n)$ eine Hypothese mit $m \geq n$, $P(X_1, \ldots, X_m)$ die Konjunktion der Prämissen $p_1, \ldots, p_m$, die die Termvariablen $X_1, \ldots, X_m$ verwenden, und $q(X_1, \ldots, X_n)$ die Konjunktion mit den Termvariablen $X_1, \ldots, X_m$. Dann gilt:*

$$
\begin{array}{lcl}
|concl(h)| & = & SELECT\ COUNT(*)\ FROM\ tab(p)\ WHERE\ cond(q); \\
|negconcl(h)| & = & SELECT\ COUNT(*)\ FROM\ tab(p)\ WHERE\ cond(not(q)); \\
|pos(h)| & = & SELECT\ COUNT(*)\ FROM\ tab(p), tab(P)\ WHERE\ cond(P,q); \\
|neg(h)| & = & SELECT\ COUNT(*)\ FROM\ tab(p), tab(P)\ WHERE\ cond(P,not(q));
\end{array}
$$

Man kann nun das Bewertungs- und Beschneidungskriterium genauso anwenden, wie unter RDT. Jedoch hat man nur die in Abschnitt 6.5.2 aufgeführten Mengen zur Verfügung.

Anmerkung zum Hypothesenraum

Die Größe des Hypothesenraumes hängt nicht von der Anzahl der Tupel in der Datenbank ab, sondern vielmehr von der Anzahl der Regelschemata, der Anzahl der Prädikate und der maximalen Anzahl der Literale in einem Regelschema. [54] grenzt die Größe ausführlich beschrieben ab.

6.5.3 Algorithmus

Die grundlegenden Gedanken von RDT können selbstverständlich erhalten bleiben. Modifikationen sind lediglich bei der Bildung der Prädikate mittels Datenbankinformationen und Abbildungen nötig, sowie beim Hypothesentest, der auf der Datenbank durchgeführt werden muß. Besondere Beachtung fällt dabei auf die richtige Behandlung der Datentypen, also der Kompatibilität zwischen den Datentypen der Datenbank und der (Prädikaten-)logik (Sorten).

6.6 RDT/DM

RDT/DM ist eine Weiterentwicklung von RDT/DB im Rahmen einer Diplomarbeit [54]. Es wurde versucht, durch das verstärkte Ausnutzen von Features eines Datenbankmanagementsystems ein verbessertes Laufzeitverhalten zu erreichen.
Vor allem wurde eine Optimierung der SELECT-Anweisungen angestrebt. Das Ergebnis einer SELECT-Anweisung wird nach der Verwendung in RDT/DB direkt wieder verworfen. Daher versucht man in RDT/DM, das Ergebnis einer Anfrage durch den Einsatz von MATERIALIZED VIEWS für weitere, zukünftige Anfragen festzuhalten.
RDT/DB nutzt keine Fremdschlüsselbeziehungen zwischen zwei Relationen aus. In RDT/DM wurden diese mit in den Algorithmus einbezogen. Leider stellte sich nach der Implementierung des Verfahrens heraus, daß die ursprüngliche Variante (RDT/DB) weiterhin ein besseres Laufzeitverhalten als die Nachfolgervariante hat. Man kann sich dies z.B. so erklären, daß das verwendete Datenbanksystem intern SELECT- Anweisungen besser zwischenspeichert, als wenn der Benutzer eine direkte Implementierung über Views wählt.

Auf den genauen Ablauf des Algorithmusses soll hier nicht weiter eingegangen werden.

6.7 Einordnung der Arbeit

Während der Seminarphase der Projektgruppe wurden andere Vorträge des Themenbereiches Wissensentdeckung in Datenbanken gehalten. Zu diesen soll in diesem Abschnitt ein Bezug hergestellt werden, um die Relevanz von RDT besser beurteilen zu können.
RDT ordnet sich bedingt durch die Weiterentwicklung zu RDT/DB direkt in den Wissensentdeckungsprozeß von KDD ein (vgl. Kapitel 2). Hat man sich bei der Auswahl des Analyseverfahrens im KDD Prozeß für RDT/DB entschieden, so wird man den Folgeschritt, das Festlegen der Verfahrensparameter, sicherlich mehrmals durchführen müssen. Man muß zunächst sinnvolle Regelschemata finden, die auf die entsprechende Datenbank anzuwenden sind. Weiterhin sollte man bereits im ersten Schritt des KDD Prozesses den möglichen Einsatz von RDT berücksichtigen.
Die Daten sollten z.B. durch Merkmalsselektion und Merkmalsgenerierung aufbereitet werden, um dem Lernverfahren eine möglichst gute Lernumgebung zu ermöglichen (vgl. Kapitel 12). Zum einen sollten unwichtige Daten durch Selektion vermieden werden, da die lange Laufzeit eine große Schwäche des Algorithmus ist. Zum anderen sollten weitere Merkmale generiert werden, da RDT/DB nur auf Attributen einer Tabelle arbeiten kann und bisher keine Operationen unterstützt werden, um diese während des Lernlaufes zu manipulieren. Eine direkte Merkmalsselektion nur durch den Einsatz von RDT/DB ist nur durch eine Anpassung der Abbildungen, welche die Attribute einer Tabelle auf die Prädikate abbilden, möglich. Die geschickte Konstruktion von Abbildungsfunktionen gestattet so auch eine Merkmalsgenerierung in gewissem Maße. Es ist jedoch im Hinblick auf den operationalen Einsatz von RDT in einer Analysekette sinnvoller, die typischen Abbildungstypen zu nutzen und zuvor die Daten mit anderen Operatoren aufzubereiten.
Weiterhin ist es anzuraten, die Verteilung der Attribute zu analysieren. Dies kann z.B. durch Erstellen eines Verteilungsgraphen geschehen. Es besteht dann die Möglichkeit, daß man bekannte statistische Verteilungen erkennt, so daß man auf einer Stichprobe der Daten weiter lernen kann, was einen enormen Geschwindigkeitsvorteil mit sich bringt. Statistische Grundlagen sollten dem Benutzer von RDT geläufig sein. Sie werden in Kapitel 3 dieser Arbeit erläutert.
RDT ist das einzige Regellernverfahren, welches während der Seminarphase vorgestellt wurde. Somit unterscheidet es sich grundlegend von anderen Lernverfahren, wie der SVM, die z.B. nicht in der Lage ist auf mehreren relational verknüpften Tabellen mit nominalen Daten zu arbeiten (vgl. Kapitel 5). Bei

RDT hat der Benutzer die Möglichkeit das Lernergebnis interaktiv durch Vorgabe von Regelschemata zu beeinflussen. Diese Art der Interaktion existiert bei keinem anderen der vorgestellten Lernverfahren.
Die Klassifikationsverfahren wie Clustering (Kapitel 10) und Subgruppenentdeckung (Kapitel 7) haben vollkommen andere Lernaufgaben als RDT. So versucht man beim Clustering möglichst homogene Gruppen einer Datenmenge aufzufinden, wobei die Objekte unterschiedlicher Gruppen möglichst heterogen sein sollen.
Es wurden zudem Verfahren vorgestellt, die auf Beispielen lernen, wie z.B. die Entscheidungsbäume in Kapitel 4. RDT ist dagegen als eine geordnete Suche im Hypothesenraum aufzufassen und somit grundlegend verschieden.
Bei der Betrachtung der Ergebnisse vergangener KDD Cups wurden Statistiken gezeigt, daß RDT eher selten zum Einsatz kam. Eine Siegerlösung setzte zudem RDT nie ein (vgl. Kapitel 13).

Kapitel 7

Subgruppenentdeckung

Lars Michele

Die Subgruppenentdeckung ist ein deskriptives Lernverfahren mit dem man versucht, lokale Aussagen über Teilmengen einer Gesamtmenge zu erhalten. Der Unterschied zu anderen Lernverfahren ist also, daß man bei der Subgruppenentdeckung kein globales Modell erstellt, sondern man erstellt mehrere lokale Modelle, wobei man nicht für jede mögliche Teilmenge ein Modell erstellt.

7.1 Was sind Subgruppen?

7.1.1 Erklärung an Beispielen

Um die Entdeckung von Subgruppen zu motivieren, betrachtet man am besten Beispiele. In der Wirtschaft benutzt man vor allem im Bereich des Marketings Subgruppenentdeckung, um z.B. Aussagen über das Kaufverhalten, Trends und Werbezielgruppen zu erhalten.
Bei einer Versicherung werden z.B. verschiedene Daten über Kunden gespeichert, die die abgeschlossenen Verträge beschreiben, aber auch allgemeine Daten wie Alter, Wohnort usw. Mit der Subgruppenentdeckung versucht man nun, lokale Beobachtungen zu machen, die für die Subgruppe gelten, aber von dem Durchschnitt der Kunden abweicht. So könnte man zum Beispiel folgende Beobachtungen machen:

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

Oder

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

An diesen Beispielen sieht man direkt, daß diese Aussagen nicht für alle Kunden gelten, aber man kann anhand dieser Aussagen Geschäftsstrategien entwickeln, z.B. bei welchen Leuten man mehr Werbung machen sollte, oder um welche Kunden man sich besonders kümmern sollte.

7.1.2 Formale Definition und Lernaufgabe

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

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

Sei weiterhin

$$S \subseteq X$$

eine gegebene, gemäß D gezogene Stichprobe der Gesamtpopulation.

Es sei schließlich q eine Funktion

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

Die Lernaufgabe Subgruppenentdeckung kann dann auf zwei Arten definiert werden:

1. Gegeben X, S, L_H, q und eine Zahl $q_{\min} \in \mathbb{R}$, finde alle $h \in L_H$, für die $q(h) \geq q_{\min}$.

   Oder/Und

2. gegeben X, S, L_H, q und eine natürliche Zahl $k \geq 1$, finde eine Menge $H \subseteq L_H$, $|H| = k$ und es gibt keine $h \in H$, $h' \in L_H \backslash H : q(h') \geqq (h)$.

Eine der wichtigsten Sachen in dieser Definition ist die Qualitätsfunktion q, die bewertet, wie interessant" eine bestimmte Hypothese bzw. die durch h repräsentierte Subgruppe $ext(h)$ ist.

7.2 Qualitätsfunktionen

Qualitätsfunktionen sollen dazu dienen, die Interessantheit von Aussagen zu bewerten. Um dies umzusetzen müßte man im besten Fall eine umfassende Benutzermodellierung machen, die die Ziele, das Interesse und das Vorwissen des Benutzers erfaßt. Mit dieser Herangehensweise sollte z.B. verhindert werden, daß bekannte Aussagen ausgeschlossen werden, da diese für die meisten Benutzer nicht besonders interessant wären. Da diese Herangehensweise aber schwierig zu bewerkstelligen ist, wird die Interessantheit von Subgruppen meistens anhand von leicht objektivierbaren statistischen Eigenschaften gemessen. Bei einer Versicherungsgesellschaft interessiert man sich zum Beispiel für eine statistisch auffällige" Subgruppe in der die Anzahl der abgeschlossenen Lebensversicherungen sich signifikant von der Verteilung in der gezogenen Stichprobe bzw. der gesamten Population unterscheidet. Desweiteren spielt auch die Größe der Subgruppe eine entscheidene Rolle, je nach Fragestellung könnten zu kleine bzw. zu große Subgruppen nicht von Interesse sein.
Wie kann man nun "statistisch auffällig" definieren? Diesen Begriff kann man am Einfachsten über Hypothesentests definieren. Der einfachste Fall wäre es, wenn man an Subgruppen interessiert ist, die in Bezug auf die Verteilung eines binären Attributs auffällig sind. Bei der Versicherungsgesellschaft könnte z.B. das Attribut "Lebensversicherung abgeschlossen?" interessant sein.
Dieses Attribut sei nun mit der Wahrscheinlichkeit p in der Gesamtpopulation verteilt. Um die statistische Auffälligkeit einer durch eine Hypothese h beschriebenen Subgruppe der Größe $n := |ext(h)|$ zu bewerten, betrachtet man die Wahrscheinlichkeit p_h, bei Ziehung eines Objektes aus dieser Subgruppe ein Objekt mit der gesuchten Eigenschaft zu erhalten. Statistisch nicht auffällig sind dann Subgruppen, die sich in Bezug auf die gesuchte Eigenschaft nicht anders verhalten als irgendeine zufällig aus der Gesamtpopulation gezogene Stichprobe der Größe n. Dies ist dann die möglicherweise zu verwerfende *Nullhypothese* über die Subgruppe h. Gilt die Nullhypothese, so ist $p_H = p$, und das Ziehen einer Subgruppe der Größe n ist die n-fache Wiederholung eines Experiments, bei dem mit der Wahrscheinlichkeit p ein Objekt mit der gesuchten Eigenschaft gezogen wird. Die relative Häufigkeit solcher Objekte in dieser Stichprobe, also in $ext(h)$, ist dann also ein geeigneter Schätzer für p:

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

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

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

In den einzelnen Stichproben der Größe n schwankt der Wert des Schätzers je nach den für die jeweilige Stichprobe ausgewählten Objekten. Was kann man aus dem in einer einzelnen Stichprobe der Größe n, nämlich der Subgruppe h, beobachteten Wert von $\hat{p}$ über die Gültigkeit der Nullhypothese schließen? Man betrachtet dazu die Wahrscheinlichkeit, mit der ein bestimmter Wert von $\hat{p}$ auftritt, falls p tatsächlich die zu Grunde liegende Wahrscheinlichkeit der gesuchten Eigenschaft ist. Je unwahrscheinlicher ein beobachteter Wert ist, desto weniger plausibel ist es, daß tatsächlich p die zugrundeliegende Wahrscheinlichkeit war, und desto naheliegender ist es, die Nullhypothese zu verwerfen, und anzunehmen, daß die betrachtete Subgruppe sich tatsächlich von der Gesamtpopulation statistisch unterscheidet. Betrachtet man als Beispiel eine Population, in der ein gesuchtes Merkmal mit der Wahrscheinlichkeit $p = 0,5$ auftritt. Bei Ziehung einer Subgruppe der Größe $n = 2$ erwartet man im Mittel ein Objekt mit der gesuchten Eigenschaft; die Wahrscheinlichkeit, zwei Objekte mit dieser Eigenschaft anzutreffen, ist jedoch immer noch $P(\hat{p} = 1|p = 0,5; n = 2) = 0,5 \cdot 0,5 = 0,25$.
Wenn man in einer Subgruppe der Größe 2 diese Verteilung beobachtet, ist dies also kein besonders starker Hinweis darauf, daß die Subgruppe sich von der Gesamtpopulation unterscheidet. Die sich für allgemeines n ergebende Wahrscheinlichkeitsverteilung $P(\hat{p} = x|p, n)$ der Werte von $\hat{p}$ (genauer die Verteilung der Anzahl der gezogenen Objekte mit der gesuchten Eigenschaft) ist die Binomialverteilung. Für genügend große Werte von n ($n > 30$ wird allgemein akzeptiert) wird diese Verteilung genügend genau durch eine Normalverteilung mit Mittelwert $\mu = p$ und Standardabweichung $\sigma = \sqrt{\frac{p \cdot (1-p)}{n}}$ genähert.
Um mit der Normalverteilung besser arbeiten zu können, führt man noch eine z-Transformation durch, die aus einer beliebigen normalverteilten Variable eine standard-normalverteilte Variable ($\mu = 0$ und $\sigma = 1$) erzeugt. Dies leistet hier die Variable V:

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

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

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

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

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

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

Daraus erhält man die Qualitätsfunktion

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

Die Qualitätsfunktion q, so wie sie definiert wurde, ordnet also die Subgruppen nach absteigender Signifikanz an. Die besten k Subgruppen dieser Liste müssen trotzdem keine Subgruppen sein, bei denen man die Nullhypothese mit genügender Sicherheit ausschließen kann. Um dies sicherzustellen, muß man ein Mindestqualitätsniveau $q_{\min}$ vorgeben, das sich gemäß den obigen Umformungen aus dem gewünschten Signifikanzniveau des statistischen Hypothesentests ergibt.

7.3 Effiziente Suche von Subgruppen

Da die Verfahren zur Entdeckung von Subgruppen nicht nur eine Lösung finden, sondern die Menge aller Lösungen oberhalb eines bestimmten Qualitätsniveaus bzw. die Menge der k besten Lösungen im Hypothesenraum finden müssen, ergeben sich einige Probleme. Ein Problem ist, daß die Verfahren keine Hill-climbing-Suche durchführen können, sondern den Hypothesenraum möglichst vollständig durchsuchen müssen. Dies kann im einfachsten Fall z. B. mit einer absteigenden Breitensuche in L_H geschehen (Abbildung 7.1). Man beginnt also mit der allgemeinsten Hypothese in diesem Raum, und verfeinert

diese dann zu immer kleineren Subgruppen. Schon bei einfachen Problemen, bei denen die Instanzen nur durch wenige Merkmale beschrieben sind, ergibt sich jedoch ein so großer Hypothesen- und damit Suchraum, daß es notwendig ist, Kriterien zu finden, mit denen möglichst große Teile des Suchraumes abgeschnitten und damit von der Betrachtung ausgeschlossen werden können. Am leichtesten ist dies möglich, wenn man für die Problemstellung relevante Eigenschaften der Hypothesen finden kann, die sich entlang eines absteigenden Suchpfades monoton entwickeln, also z. B. niemals größer, sondern nur kleiner werden können. Sobald bei einer absteigenden Suche diese Größe einen geforderten Mindestwert unterschreitet, können alle darunterliegenden Hypothesen als Lösung ausgeschlossen werden. Für die bei der Subgruppenentdeckung relevante Eigenschaft, die Qualität q, gilt leider eine solche Monotonieeigenschaft im allgemeinen Fall nicht. Je nachdem, wie sich die Verteilungen bei einer Verkleinerung der Subgruppe ändern, kann die Qualität der Subgruppe fallen oder auch steigen.
Für die zweite Variante des Subgruppenentdeckungsproblems, das Finden der k besten Subgruppen, kann man dennoch in vielen Fällen große Teile des Suchraumes abschneiden, indem man sich sogenannter optimistischer Schätzfunktionen für q bedient [70]. Das Ziel einer optimistischen Schätzfunktion ist es, gegeben eine bestimmte Hypothese h und ihre Qualität $q(h)$, abzuschätzen, wie sich die Qualitätsfunktion q bei einer weiteren absteigenden Suche ausgehend von h günstigstenfalls entwickeln könnte. Die optimistische Schätzfunktion sollte garantieren, daß für keine der unterhalb von h liegenden spezielleren Hypothesen der Wert von q oberhalb der optimistischen Schätzung liegt. Kann man dies zusichern, so kann man immer dann den Teilraum unterhalb einer Hypothese h abschneiden, wenn die optimistische Schätzung für diesen Teilraum unterhalb der Qualität der schlechtesten der bisher gefundenen k Hypothesen liegt.
Man kann dies wie folgt präzisieren. Sei $\rho : L_H \rightarrow 2^{L_H}$ der Spezialisierungsoperator, der bei der absteigenden Suche zu einer Hypothese h alle unmittelbar spezielleren Nachfolger $\rho(h)$ liefert, und sei ρ^* die transitive Hülle von ρ, also der gesamte Teilraum unterhalb einer Hypothese h. Eine optimistische Schätzfunktion für q ist dann eine Funktion:

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

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

proc Subgruppen$(S, \rho, q, q_{min}, g_{min}, k)$
$Q :=$ {„Ganze Population"}, $H := \emptyset$

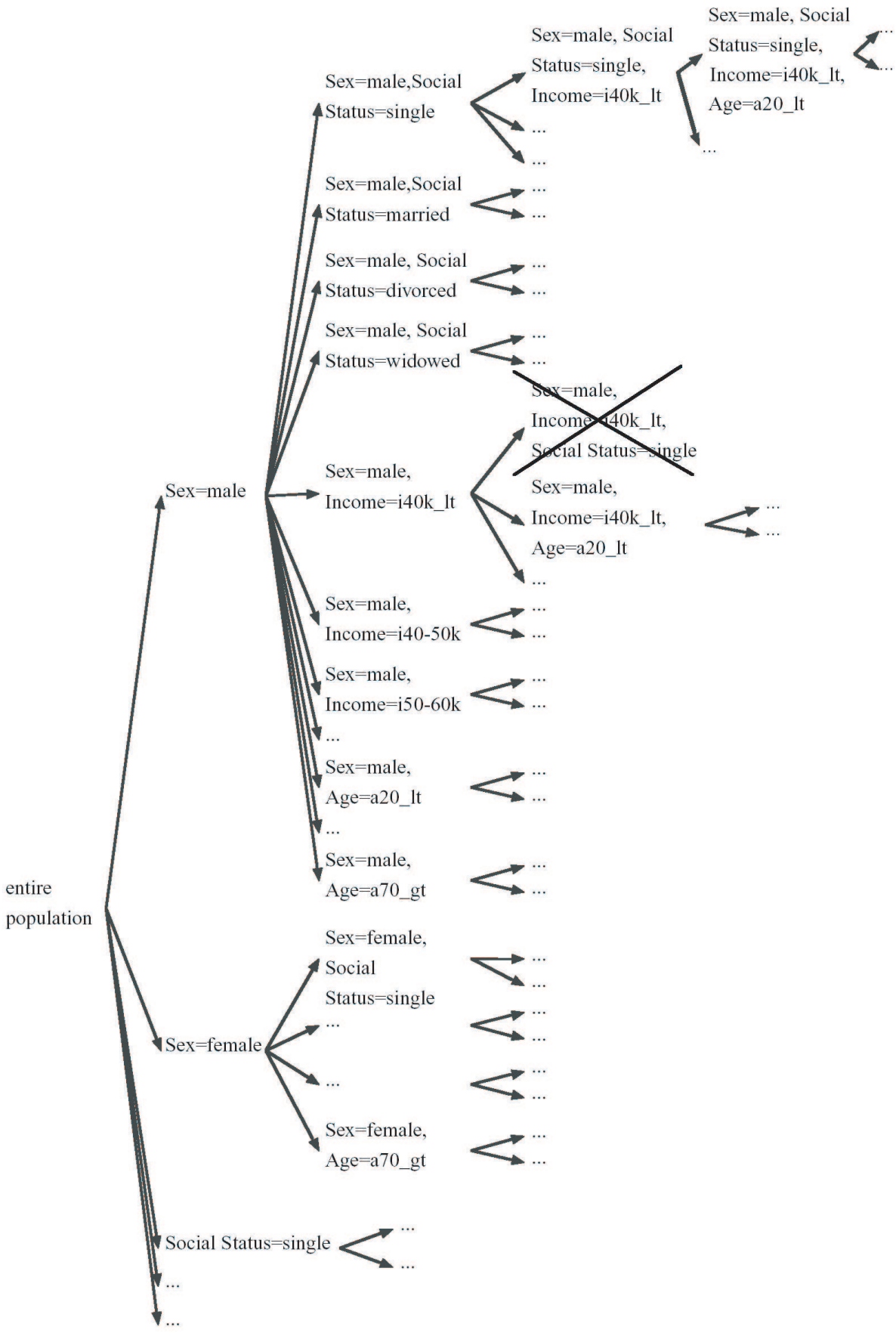


Abbildung 7.1: Suchbaum bei der propositionalen Subgruppensuche

while $Q \neq \emptyset$
  Wähle $C \subseteq Q$ (gemäß Suchstrategie)
  $\rho(C) := \bigcup_{h \in C} \rho(h)$
  forall $h \in \rho(C)$
    Teste h auf S (berechne $q(h)$)
    if $q(h) \geq q_{min}$ und $g(h) \geq g_{min}$ **then**
      if $q(h) > min_{h \in H} q(h)$ **then**
        if $|H| < k$
        then $H := H \cup \{h\}$
    if $q_{max(h)} < min_{h \in H} q(h)$ oder $g(h) < g_{min}$
      then (Teilbaum wird abgeschnitten)
      else $Q := Q \cup \{h\}$
return H

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

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

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

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

7.4 Erweiterungen von Subgruppen

7.4.1 Mächtigere Hypothesenräume

Die beschriebene Subgruppensuche kann selbstverständlich nicht nur für Hypothesenräume durchgeführt werden, die sich auf Merkmalsvektoren beziehen, sondern für jeden Hypothesenraum, in dem sich ein Verfeinerungsoperator definieren läßt. So gibt es beispielsweise Subgruppenverfahren für multirelationale prädikatenlogische Hypothesenräume, bei denen durch geeignete Sprachbeschränkungen sichergestellt wird, daß der entstehende Suchraum handhabbar bleibt. Hier haben sich Einschränkungen bewährt, die an die Fremdschlüsselbeziehungen in Datenbanken angelehnt sind [70]. Dies wird in Abschnitt 7.5 noch genauer erörtert.

7.4.2 Zeitbezogene Subgruppenmuster

Bei vielen Marktuntersuchungen ist es üblich, die Untersuchungen in bestimmten Abständen zu wiederholen. So könnte beispielsweise ein Unternehmen eine jährliche Kundenbefragung durchführen und daran interessiert sein, die Gesamtheit der Fragebogenrückläufe auch im zeitlichen Trend mit Subgruppenverfahren zu untersuchen. Es sind zu diesem Zweck spezielle Qualitätsfunktionen entwickelt worden, die nicht nur einen Vergleich von Subgruppen mit einer Gesamtpopulation durchführen, sondern dies auch über mehrere in zeitlichen Abständen erhobene Teilpopulationen ("Segmente") tun können [47]. Dabei muß als technische Schwierigkeit beachtet werden, dass die Beschreibungen der Instanzen in den einzelnen Segmenten nicht notwendigerweise die gleichen Attribute aufweisen [46].

7.4.3 Optimierung von Subgruppenergebnissen

Formuliert man deskriptive Lernprobleme mit Qualitätsfunktionen, die auf die einzelnen Hypothesen bezogen sind, so ergibt sich zwangsläufig das Problem, daß die Gesamtmenge aller gefundenen Hypothesen möglicherweise in sich nicht gut aufeinander abgestimmt ist. So kann es bei der Subgruppenentdeckung ohne weitere Vorkehrungen beispielsweise vorkommen, daß man entdeckt, daß alleinstehende Männer doppelt so häufig Lebensversicherungen abschließen wie die Gesamtpopulation, und daß wir gleichzeitig als Entdeckung zurückliefern, daß alleinstehende Männer in ländlichen Gegenden doppelt so häufig Lebensversicherungen abschließen wie die Gesamtbevölkerung. Allgemein ergibt sich also die Frage, wie inhaltlich ähnliche Subgruppen erkannt und dann die weniger interessante von zwei sehr ähnlichen Subgruppen "unterdrückt" werden kann. In unserem Beispiel ist vermutlich die zweite Beobachtung weniger interessant, weil ihr Inhalt vollständig aus der ersten Beobachtung ableitbar ist. Man kann zur Unterdrückung solch redundanter Subgruppen Maße definieren, die die sogenannte *Affinität* von Subgruppen berechnen, und kann dann Schwellwerte festlegen, so daß oberhalb einer gewissen Affinität und unterhalb eines bestimmten Unterschieds in der Interessantheit die schwächere der beiden Subgruppen unterdrückt wird [34].

7.5 Multirelationale Entdeckung von Subgruppen

7.5.1 Lernaufgabe

Seien folgende Voraussetzungen gegeben:

- eine relationale Datenbank D mit den Relationen $R = \{r_1, ..., r_m\}$

- ein Hypothesenraum L_H
- eine Qualitätsfunktion $d : h \in L_H,\ D \to [0, 1]$
- eine natürliche Zahl $k > 0$

Dann ist die Lernaufgabe folgendermaßen definiert:

- finde eine Menge $H \subseteq L_H$ von Hypothesen mit weniger als k Elementen,
- so daß für alle $h \in H$, $d(h, D) > 0$ gilt
- und für alle $h' \in L_H \backslash H$, $d(h', D) \leq \min_{h \in H} d(h, D)$ gilt.

7.5.2 Hypothesensprache und Überdeckung

Um nun diese Lernaufgabe zu lösen, muß man die Hypothesensprache genauer definieren. Die Hypothesensprache muß Restriktionen für Attribute erlauben, es müssen mehrere Relationen zu einer Gruppe zusammengefaßt werden können und die Hypothesensprache muß verschiedene Restriktionen verknüpfen können.
Wenn man nun eine Subgruppe zu einer gegeben Objektrelation r_0 welche ein Zielattribut K hat und Teil der Datenbank D gegeben hat, kann man die von der Hypothese $h \in L_H$ überdeckte Subgruppe dann folgendermaßen definieren:

$$c(h) := \pi_K(\{\sigma | h\sigma \in D\})$$

Die Überdeckung $c(h)$ von h sind also alle Daten aus D, die die Hypothese h erfüllen.

7.5.3 Fremdschlüssel

Um den Hypothesenraum klein zu halten benutzt man *foreign key links* (Fremdschlüssel) die in vielen Datenbanken vorhanden sind. Ein *foreign key link* ist eine Verknüpfung von einem Attribut aus einer Relation zu einem Attribut einer anderen Relation. Relationen dürfen dann nur als Gruppen anhand dieser *foreign links* zusammengefaßt werden. Man kann also beim aufstellen einer Hypothese nur für verlinkte Attribute auch die gleichen Variablen nehmen und bei einer Verknüpfung von Relationen müssen die Relationen mit mindestens einer anderen Relation ein Literal teilen.
Da bei *foreign key links* Zyklen auftreten können limitiert man die Anzahl der Verknüpfungen von Relationen. Diese Schranke ist üblicherweise der längste mögliche Pfad ohne Zyklus.

7.5.4 Ein optimaler Spezialisierungsoperator

Der optimale Spezialisierungsoperator soll den Hypothesenraum präzise erfassen. Sei $F = \{f_1, ..., f_n\}$ eine Menge von Foreign (Key) Links aus der Datenbank D mit den Relationen $R = \{r_1, ..., r_m\}$. Wenn es mehrere Links gibt, die von der Relation r starten, so sollen sie nach Nummern σ_r geordnet sein.
Definition (Kompatibilität von Variablen): Sei $C = L_1, ..., L_n$ eine Menge von Literalen, V eine Variable die als erstes in L_i auftaucht mit einem Funktor r und an Position p_1 und U eine andere Variable die als erstes in L_j auftaucht mit einem Funktor s an Position p_j. V und U sind kompatibel unter Berücksichtigung von C und einer Menge von Links $F(V \bowtie_{C,F} U)$ wenn gilt: $r[p_i] \rightarrow s[p_j] \in F$. Für zwei Mengen von Variablen $\mathbf{V}$ und $\mathbf{U}$ definiert man: $\mathbf{V} \bowtie_{C,F} \mathbf{U} := \{(V, U) \in \mathbf{V} \times \mathbf{U} | \; V \bowtie_{C,F} U\}$.
Den Spezialisierungsoperator für L_H kann man nun wie folgt definieren. Sei $h := L_1, ..., L_n$ mit den Variablen $\mathbf{V} := vars(h)$ die zu spezialisierende Hypothese, dann ist der Spezialisierungsoperator ρ wie folgt definiert:

1. Für alle L_i in h, $h' := L_1, ..., L_{i-1}, \rho(L_i), L_{i+1}, ..., L_n$ die in $\rho(h)$ sind setze $[o = i.1]$.

2. Für alle $L_i = r(V_1, ..., V_{a(r)})$ in h, so daß $r[m] \rightarrow s[k] \in F$ ist, sei $\mathbf{U} = \{U_1, ..., U_{k-1}, U_{k+1}, ..., U_{a(s)}\}$ eine Menge neuer Variablen, also $vars(h) \cap \mathbf{U} = \emptyset$. Dann gilt, daß

$$h' := \bigwedge^{L_1,...,L_n,s(U_1,...,U_{k-1},V_m,U_{k+1},...,U_{a(s)}),}_{U \in \mathbf{U}} any(U) \bigwedge_{(V,U) \in \mathbf{V} \bowtie \mathbf{U}} any(V,U)$$

in $\rho(h)$ ist. Setze $[o = i.2.o_r(r[m] \rightarrow s[k])]$.
Der Spezialisierungsoperator für ein einzelnes Literal L wird dann wie folgt definiert:

1. $L = any(X)$, X ist mit den Werten $\{v_1, v_2, ..., v_{n-1}, v_n\}$ geordnet. Falls $n = 2$ ist, sind $X = v_1$ und $X = v_n$ in $\rho(L)$ ansonsten sind $X \geq v_2$ und $X \leq v_{n-1}$ in $\rho(L)$.

2. $L = X \geq v$ $(L = X \leq v)$, X ist mit den Werten $\{..., u, v, w, ...\}$. Falls $w = v_n$ $(u = v_1)$ ist, dann sind $X = v$ und $X = w$ $(X = u)$ in $\rho(L)$, ansonsten sind $X = v$ und $X \geq w$ $(X \leq u)$ in $\rho(L)$.

3. $L = any(X)$ $(L = X = a)$, X ist baumartig strukturiert, wobei b_1 bis b_n die Kinder von any(von a) im Wertebaum von X sind. Dann sind $X = b_1, ..., X = b_n$ in $\rho(L)$.

4. $L = any(X, Y)$. Dann ist $X = Y$ in $\rho(L)$.

In den eckigen Klammern ist die Ordnung von ρ spezifiziert. Für jede Hypothese h können nun zwei speziellere Hypothesen $h_1 \in \rho(h)$ und $h_2 \in \rho(h)$ nach der lexikographischen Vergleich zwischen $o(h_1)$ und $o(h_2)$ sortiert werden, also kann ganz $\rho(h)$ geordnet werden. Der optimale Spezialisierungsoperator ρ_{opt} ist dann

$$\rho_{opt} := \{h' \in \rho(h) \mid o(h') > o(h)\}.$$

Durch diese Definition von ρ_{opt} wird für jede Hypothese genau ein Verfeinerungspfad erzeugt, womit verhindert wird, daß mehrere redundante speziellere Hypothesen erzeugt werden. Diese Eigenschaft ist vor allem für die Parallelisierung des Algorithmus wichtig.

7.5.5 Die Qualitätsfunktion

Sei nun eine Relation r_o mit dem Key-Attribut K gegeben, die zu einer zu untersuchenden Datenbank D gehört. Im einfachsten Fall hat man wieder ein binäres Zielattribut A_g in r_o. Sei $T := \{t \in r_o | r_o[A_g] = 1\}$ die Menge der zu findenden Zielobjekttupel, $g(h) := \frac{|c(h)|}{|r_o|}$ die Generalität einer Hypothese h und $p_0 := \frac{T}{r_o}$ und $p(h) := \frac{c(h) \cap T}{c(h)}$ seien Wahrscheinlichkeiten. Die Evaluierungsfunktion wird dann folgendermassen definiert:

$$d(h) = \begin{cases} 0 \ if \ |c(h)|/|r_0| > s_{min} \\ g(h)(p(h) - p_0) \ sonst \end{cases}$$

Der erste Teil der Funktion beschreibt die Mindestgröße, die eine Subgruppe haben soll, der zweite Teil der Funktion ist die Standardevaluierungsfunktion, die schon weiter vorne vorgestellt wurde.

7.5.6 Der MIDOS-Algorithmus

Wenn man all diese Definitionen hat, ist der MIDOS-Algorithmus relativ einfach:

Sei $Q := r_o(V_1, ..., V_{a(r_o)})$ und $H := \emptyset$.
While $Q \neq \emptyset$

- wähle eine Teilmenge C aus Q gemäß der gegebenen Suchstrategie
- sei $\rho(C) := \{\rho(h) | h \in C\}$
- teste jedes $h \in \rho(C)$ auf D (führe $d(h, D)$ aus)
    - if $d(h, D) = 0$, schneide den Teilbaum ab.

- else if $d_{\max}(h, D) < \min_{h' \in H} d(h', D)$, schneide den Teilbaum ab.
- else

* if $d(h, D) > \min_{h' \in H} d(h', D)$, entferne das kleinste Element aus H und füge h hinzu

* füge h zu Q hinzu

7.6 Zusammenfassung

Als deskriptives Lernverfahren ist die Subgruppenentdeckung sehr mächtig. Die größten Probleme liegen in der Größe des Hypothesenraumes, sprich in der Anzahl der vorhandenen Attribute, und in der Beschaffenheit der Qualitätsfunktion. Um diese Verfahren effizient zu machen wird einmal der optimale Spezialisierungsfaktor benutzt, um redundanzen bei den Subgruppen zu vermeiden, zum anderen wird versucht mit Hilfe der Wahrscheinlichkeitsrechnung und Statistik optimistische Schätzfunktionen zu benutzen.
Im KDD-Prozess kann die Subgruppenentdeckung einmal zum Erwerb von Datenverständnis oder als Lernverfahren für lokale Aussagen eingesetzt werden. Um die Subgruppenentdeckung anzureichern, kann man zusätzlich noch vorher mit Merkmalsgenerierung und Selektion versuchen, den Hypothesenraum zu verkleinern bzw. verständlicher zu machen.
Im Vergleich mit anderen Lernverfahren ist die Subgruppenentdeckung dem Aprioriverfahren sehr ähnlich, wenn man nur Subgruppen mit einer bestimmten Mindestgröße ausgibt. Einziger Uunterschied dann ist, daß die Subgruppenentdeckung ein Zielattribut benutzt und somit überwachtes Lernen ist, während man beim klassischen Aprioriverfahren unüberwachtes Lernen hat und dadurch das Lernergebnis stark abweichen kann. Lernverfahren wie Bagging und Boosting, SVM's, Entscheidungsbäume und RDT's versuchen ein globales Modell zu bilden und verfolgen also ein komplett anderes Ziel als die Subgruppenentdeckung. Clustering ist der Subgruppenentdeckung zwar ähnlich, sucht aber sich nicht überschneidende Subgruppen die den ganzen Hypothesenraum abdecken um auch ein globales Modell zu erstellen und ist auch wie das Aprioriverfahren ein unüberwachter Lerner.
Ein großer Unterschied der Subgruppenentdeckung zu anderen Verfahren ist noch die Möglichkeit auch auf relationalen Daten zu arbeiten zu arbeiten und nicht wie die meisten Lernverfahren nur auf einer Tabelle.

Kapitel 8

Apriori

Dirk Dach

8.1 Einleitung

Fortschritte bei der Datengewinnung und -speicherung haben dazu geführt, dass bei Unternehmen große Datenbanken mit Verkaufsdaten aufgebaut wurden. Eine der größten Herausforderungen für Unternehmen ist, aus diesen Daten nutzbares Wissen zu gewinnen. In diesem Text geht es um die Mustererkennung in Datenbanken und insbesondere um das bekannteste Problem in diesem Bereich, dem Finden von Assoziationsregeln. Als Grundlage für die Beschreibung des Apriori Verfahrens bzw. der DFS-Algorithmen dienten die Artikel [36] bzw. [38]. Ein anschauliches Beispiel für Assoziationsregeln stellt die Analyse von Transaktionen in Supermärkten dar. Scannerkassen ermöglichen die genaue Erfassung aller Transaktionen. Man will nun das Kundenverhalten im Bezug darauf, welche Produkte zusammen gekauft werden, untersuchen. Das Problem ist aber auch für viele andere Bereiche relevant, z.B Onlineshops und Versandhändler. Diese Regeln helfen dem Unternehmen, Entscheidungen über Marketingstrategien, Mailing-Aktionen, Cross-Marketing, Preisgestaltung, Produktplazierung oder Katalogdesign zu treffen. Ein einfaches Beispiel für eine solche Regel ist Bier⇒Chips(80%). Sie besagt, daß 4 von 5 Kunden, die Bier gekauft haben, gleichzeitig Chips kaufen. Für die Produktplazierung im Supermarkt wäre von Vorteil, Chips in der Nähe von Bier zu platzieren.

8.2 Problembeschreibung

Im folgenden wird das Problem formal beschrieben.

Sei A die Menge von Artikeln, die der Supermarkt führt. $X = \{i_1, \ldots i_k\} \subseteq A$ wird Artikelmenge(„Itemset“) genannt. Wenn X genau k Artikel enthält, ist X eine k-Artikelmenge.
Eine Transaktion über A ist ein Paar T=(tid,I), wobei tid die Transaktionsnummer und I eine Artikelmenge ist. Die Transaktionsdatenbank TD ist die Menge aller Transaktionen.
Das Cover einer Artikelmenge X ist die Menge, die aus allen Transaktionnummern von Transaktionen in TD besteht, die X enthalten. Formal wird dies folgendermaßen dargestellt:

$$cover(X, TD) := \{tid | (tid, I) \in TD, X \subseteq I\}.$$

Der Support einer Artikelmenge X in TD ist die Anzahl der Transaktionen im Cover von X in D:

$$support(X, TD) := |cover(X, TD)|.$$

Da die leere Menge in allen Transaktionen enthalten ist gilt, daß der Support der leeren Menge der Mächtigkeit von TD entspricht: $|TD| = support(\{\}, D)$
Eine Artikelmenge wird “häufig“ genannt, falls sie nicht unter einer vorgegebenen minimalen Supportschwelle s_{min} liegt. Der Support ist ein absolutes Maß. Es gibt aber auch noch ein relatives Maß, die Frequenz einer Artikelmenge. Diese gibt die Wahrscheinlichkeit an, daß X in einer Transaktion in D vorkommt. Die minimale Supportschwelle ist hierbei eine Zahl zwischen Null und Eins:

$$frequency(X, TD) := P(X) = \frac{support(X,TD)}{|D|}.$$

Mit der Transaktionsdatenbank TD und der minimalen Supportschwelle s_{min} läßt sich nun die Menge aller häufigen Artikelmengen definieren. Sie besteht aus der Menge aller Artikelmengen X, für die gilt, daß der Support von X in Transaktionsdatenbank TD größer als der Schwellenwert s_{min} ist:

$$F(TD, s_{min}) := \{X \subseteq A | support(X, TD) \geq \sigma\}.$$

Algorithmen, die Assoziationsregeln finden, gehen häufig in zwei Phasen vor. Zuerst werden alle häufigen Artikelmengen gesucht; danach werden aus den häufigen Artikelmengen die Regeln erzeugt. Das erste Problem besteht darin aus der Transaktionsdatenbank TD über der Menge von Artikeln A, die Menge der häufigen Artikelmengen $F(TD, s_{min})$ unter Berücksichtigung des Schwellenwertes s_{min} zu bestimen(„Itemset Mining“). Außerdem soll für jede häufige Artikelmenge der genaue Wert des Supports ermittelt werden. Wenn die häufigen Artikelmengen bestimmt wurden, können die Assoziationsregeln abgeleitet werden.

Eine Assoziationsregel hat die Form X $\Rightarrow$ Y, wobei X und Y Artikelmengen sind. Außerdem muss gelten, daß die Mengen X und Y disjunkt sind, d.h es darf kein Artikel sowohl in X als auch in Y vorkommen. Die Bedeutung der Regel ist folgende: Falls alle Artikel, die in X vorkommen in der Transaktion T enthalten sind, sind auch alle in Y vorkommenden Artikel in T enthalten. Der erste Teil der Regel X wird mit "Prämisse", der zweite Teil Y mit "Konklusion" bezeichnet. Auch für Assoziationsregeln gibt es einen Support. Damit die Assoziationsregel gilt müssen logischerweise alle Artikel der Regel in der Transaktion vorkommen. Also entspricht der Support der Regel $X \Rightarrow Y$ dem Support der Artikelmenge $X \cup Y$. Eine Regel ist häufig, wenn ihr Support den vorgegebenen Schwellenwert s_{min} erreicht. Zusätzlich wird bei den Assoziatonsregeln noch die Konfidenz berechnet. Hierbei handelt es sich um die bedingte Wahrscheinlichkeit, daß die Artikelmenge Y in einer Transaktion T enthalten ist, unter der Voraussetzung, daß die Artikelmenge X in T enthalten ist:

$$confidence(X \Rightarrow Y, TD) := P(X|Y) = \frac{support(X \cup Y, TD)}{support(X, TD)}.$$

Es wird also das Verhältnis gebildet zwischen den Transaktionen, die X **und** Y enthalten und den Transaktionen, die X enthalten, aber nicht unbedingt Y. Falls ein vorgegebener minimaler Sicherheitsschwellwert c_{min} erreicht, wird die Regel als konfident bezeichnet. Nun kann man die Menge der häufigen und konfidenten Assoziationsregeln beschreiben. Gegeben sei eine Transaktionsdatenbank TD über einer Menge von Artikeln A, ein minimaler Supportschwellwert s_{min} und ein minimaler Konfidenzschwellwert c_{min}. Dann besteht die Menge aus allen Assoziationsregeln $X \Rightarrow Y$, bei denen X und Y disjunkt sind und für die gilt, daß $X \cup Y$ häufig und $X \Rightarrow Y$ konfident ist. Formal aufgeschrieben:

$$\begin{aligned} R(TD, s_{min}, c_{min}) \quad := \quad & \{X \Rightarrow Y | X, Y \subseteq A, X \cap Y = \{\}, \\ & X \cup Y \in F(TD, s_{min}), confidence(X \Rightarrow Y, D) \geq s_{min}\} \end{aligned}$$

Die Bestimmung dieser Menge entspricht genau dem Problem, alle Assoziationsregeln in einer Transaktionsdatenbank TD zu finden(„Association Rule Mining"). Auch hier sollen wieder die genauen Werte von Support und Konfidenz jeder Regel bestimmt werden und nicht nur die Tatsache, daß sie in der gesuchten Menge liegen. Erwähnenswert ist noch, daß das Itemset Mining Problem ein Spezialfall des Association Rule Mining Problems ist. Jede häufige Artikelmenge X entspricht nämlich der trivialen Regel $X \Rightarrow \{\}$. Da die leere Menge Teil jeder Transaktion ist, hat diese Regel eine Sicherheit von 100%.
Zur Speicherung der Transaktionsdaten werden i.d.R. binäre Datenbanken eingesetzt. Binär bedeutet, daß die Einträge nur Null oder Eins sein können.Für solche Datenbanken gibt es zwei unterschiedliche Datenbanklayouts: das horizontale und das vertikale Datenbanklayout. Beim horizontalen Datenbanklayout besteht die Datenbank aus einer Menge von Transaktionen mit den

darin enthaltenen Artikeln; beim vertikalen Datenbanklayout hingegen aus einer Menge von Artikeln mit ihren Covers. Eine Eins bedeutet, daß der Artikel in der Transaktion vorkommt bzw. die Transaktion im Cover des Artikels enthalten ist.

8.3 Ein Beispiel

Die Menge der Artikel sei $A = \{Bier, Chips, Pizza\}$ und die in Tabelle 8.1 gegebene Transaktionsdatenbank TD liege vor.

tid	Bier	Chips	Pizza
1	1	1	1
2	1	0	0
3	1	1	0
4	0	0	1

Tabelle 8.1: Beispiel einer Transaktionsdatenbank

Betrachtet man die Artikelmenge $\{Pizza, Bier\}$, sieht man, daß Pizza in zwei Transaktionen, $\{Pizza, Bier\}$ hingegen nur in einer Transaktion vorkommt. Daraus kann nun die Konfidenz berechnet werden:

$$support(Pizza \Rightarrow Bier) = support(Pizza \cup Bier, D) = 1$$
$$confidence(Pizza \Rightarrow Bier) = \frac{support(Pizza \cup Bier, D)}{support(Pizza, D)} = \frac{1}{2}$$

Für die Regel $Bier \Rightarrow Chips$ ergibt sich folgendes:

$$support(Bier \Rightarrow Chips) = support(Bier \cup Chips, D) = 2$$
$$confidence(Bier \Rightarrow Chips) = \frac{support(Bier \cup Chips, D)}{support(Bier, D)} = \frac{2}{3}$$

8.4 Der Apriori Algorithmus

Der Apriori Algorithmus berechnet alle häufigen unsicheren Assoziationsregeln. Dazu geht er in zwei Schritten vor. Zuerst werden alle häufigen Artikel bzw. Artikelmengen berechnet, danach aus der Menge der häufigen Artikel/Artikelmengen die Assoziationsregeln. In den folgenden Abschnitten werden Lösungen für beide Probleme sowie einige Optimierungsmöglichkeiten dargestellt.

8.4.1 Entdeckung häufiger Artikelmengen

Da der betrachtete Suchraum mit $2^{|I|}$ sehr groß ist, kann der naive Ansatz, alle möglichen Artikelmengen zu generieren und deren Support zu zählen, nicht zum Erfolg führen. Der Apriori Algorithmus geht daher in mehreren Durchgängen vor:

- Im ersten Durchgang wird die Datenbank nach einzelnen Artikel, d.h. einelementige, Artikelmengen, durchsucht. Es wird deren Support gezählt und die Menge der häufigen Artikel bestimmt.

- Aus den häufigen Artikeln werden potentielle zweielementige häufige Artikelmengen erzeugt. In einem weiteren Durchlauf durch die Datenbank wird bestimmt, welche davon häufig sind. Die Ergebnismenge dient dann wieder als Grundlage, um neue potentielle häufige Artikelmengen mit einem Element mehr zu erzeugen, usw. .

- Der Algorithmus bricht ab, wenn keine neuen häufigen Artikelmengen mehr gefunden werden.

Die neu erzeugten potentiell häufigen Artikelmengen werden Kandidatenmengen genannt. Der Vorteil dieser Vorgehensweise besteht darin, daß die Kandidaten nur aus den Artikelmengen erzeugt werden, die sich im vorherigen Durchgang als häufig erwiesen haben. Hierfür ist es nicht nötig die Datenbank zu benutzen. Der Grund, warum dies möglich ist, liegt in der Support Monotonie:

Support Monotonie
Sei TD eine Transaktionsdatenbank über A und $X, Y \subseteq A$ zwei Artikelmengen. Dann gilt:

$$X \subseteq Y \Rightarrow support(Y) \leq support(X)$$

Die Korrektheit dieser Behauptung ist einfach einzusehen. Wenn X eine Teilmenge von Y ist kann Y nicht häufiger als X auftreten, da jedes Vorkommen von Y gleichzeitig ein Vorkommen von X impliziert. Dies bedeutet, daß falls eine Artikelmenge häufig ist, alle Teilmengen auch häufig sein müssen. Also können Kandidatenmengen der Kardinalität k+1 erzeugt werden, indem alle häufigen Artikelmengen der Größe k um alle möglichen Artikel erweitert werden. Stellt sich für einen Kandidaten heraus, daß eine seiner Teilmengen nicht häufig ist, so kann er wieder gelöscht werden. Hierdurch wird die Anzahl der Kandidaten reduziert.
Im folgenden wird angenommen, daß die Artikel in den Artikelmengen in ihrer lexikographischen Ordnung sortiert sind. Der Algorithmus Kandidatengenerierung sieht dann folgendermaßen aus:

Erzeuge-Kandidaten(F_k)
1.) $C_{k+1} = \{\}$
2.) **for all** $X, Y \in F_k, X[i] = Y[i], 1 \leq i \leq k-1$ und $X[k] < Y[k]$ **do**
3.) $I = X \cup \{Y[k]\}$
4.) **if** $\forall J \subset I, |J| = k : J \in F_k$ *then*
5.) $C_{k+1} = C_{k+1} \cup I$
6.) **return** C_{k+1}

Als Eingabe bekommt der Algorithmus die Menge aller häufigen Artikelmengen der Größe F_k. In Zeile 2 und 3 werden die Kandidaten für die häufigen Artikelmengen der Größe C_{k+1} generiert. Anschließend wird überprüft, ob alle Teilmengen des Kandidaten häufig sind und im positiven Fall der Kandidat in die Menge C_{k+1} aufgenommen.

Mit dem Algorithmus zur Kandidatengenerierung ist es nun möglich, die häufigen Artikelmengen zu berechnen:

Erzeuge-häufige-Artikelmengen(TD, s_{min})
1.) $F_1 = \{Menge\ aller\ häufigen\ Artikel\}$
2.) **for** $(k = 2; F_{k-1} \neq \{\}; k++)$ **do**
3.) $C_k =$ Erzeuge-Kandidaten(F_{k-1})
4.) **for all** transactions $(tid, I) \in TD$ **do**
5.) **for all** $X \in C_k$ **do**
6.) **if** $X \subseteq I$ **then**
7.) X.support++
8.) $F_k = \{X \in C_k | X.support \geq s_{min}\}$
9.) return $\{F_i | 1 \leq i \leq k-1\}$

In Zeile 1 wird die Menge aller häufigen Artikel F_1 erzeugt. In der For-Schleife (Zeile 2) werden zuerst die Kandidaten der Größe k generiert. Dann werden alle Transaktionen durchlaufen und für jeden Kandidaten überprüft, ob er in der Transaktion vorkommt. Wenn das der Fall ist wird der Wert X.support um Eins erhöht(Zeilen 3-7). Alle Kanditaten, deren Support über dem Schwellenwert liegt, werden schließlich in die Menge F_k aufgenommen(Zeile 8).
Der Algorithmus terminiert, wenn für ein k keine häufigen Artikelmengen mehr gefunden wurden, also die Menge F_k leer ist. Als Ergebnis werden die häufigen Artikelmengen aller Größen zurückgeliefert.

8.4.2 Regelgenerierung

Auch hier ist der Suchraum mit $3^{|I|}$ Elementen sehr groß. Allerdings verkleinert sich der Suchraum stark, wenn die Regelgenerierung auf die häufigen Artikelmengen aufbaut. Von allen anderen Regeln weiß man, daß sie nicht häufig sind, da der Support einer Regel $X \Rightarrow Y$ dem Support der Artikelmenge $X \cup Y$ entspricht. Für jede häufige Artikelmenge $I = X \cup Y$ gibt es höchstens $2^{|I|}$ Regeln der Form $X \Rightarrow Y$. Um diesen immer noch großen Suchraum effizient zu durchsuchen, werden die Assoziatonsregeln ebenfalls in mehreren Durchgängen erzeugt. Hierzu wird eine ähnliche Eigenschaft wie die Support Monotonie ausgenutzt, die Konfidenz Monotonie.

Konfidenz Monotonie
Seien $X, Y, Z \subseteq A$ Artikelmengen, so daß $X \cap Y = \{\}$ ist. Dann gilt

$$confidence(X \backslash Z \Rightarrow Y \cup Z) \leq confidence(X \Rightarrow Y)$$

Beweis
Anwendung der Definition der Konfidenz liefert

$$\frac{support(X \cup Y \cup Z)}{support(X \backslash Z)} \leq \frac{support(X \cup Y)}{support(X)}.$$

Da $X \cup Y \subseteq X \cup Y \cup Z$ gilt wegen der Support Monotonie $support(X \cup Y \cup Z) \leq support(X \cup Y)$. Ebenso gilt $support(X \backslash Z) \geq support(X)$. Damit kann der linke Bruch nicht größer als der rechte Bruch sein.

Das bedeutet, wenn man für eine Regel $X \Rightarrow Y$ ein $Z \in X$ aus der Prämisse der Regel entfernt und in die Konklusion einfügt, kann sich die Konfidenz nur verkleinern. Sobald für eine Erweiterung der Konklusion einer Regel gilt, daß die Regel nicht mehr konfident ist, kann für keine Obermenge dieser Konklusion die Regel wieder konfident werden. Umgekehrt bedeutet dies natürlich auch, daß alle Teilmengen eine Konklusion einer konfidenten Regel auch zu einer konfidenten Regel führen. Die Konfidenz einer Regel $X \Rightarrow Y$ kann einfach berechnet werden, da der Support von X und Y schon bei der Erzeugung der häufigen Artikelmengen berechnet wurde.
Der Algorithmus zur Generierung aller häufigen und konfidenten Regeln geht ähnlich wie der Algorithmus zur Berechnung aller häufigen Artikelmengen vor und arbeitet in mehreren Durchgängen:

Generiere-konfidente-Regeln(TD, c_{min}, s_{min})
1.) Berechne die Menge der häufigen Artikelmengen.
2.) Für jede häufige Artikelmenge X

3.) $X \rightarrow \{\}$ ist eine konfidente Regel.
4.) $C_1 = \{Menge\ aller\ einelementigen\ Teilmengen\ von\ X\}$
5.) k=1
6.) Solange $C_k \neq \{\}$
7.) Erzeuge die Menge R_k, die alle Kandidaten enthält, die bei
8.) Verschieben von Prämisse nach Konklusion eine konfidente Regel
9.) ergeben.
10.) Generiere aus R_k die neue Kandidatenmenge C_{k+1}.
11.) Streiche alle Kandidaten, für die eine k-Teilmenge nicht in R_k ist.
12.) k=k+1
13.)Erzeuge aus $R_i, 1 \leq i \leq k$, die konfidenten Regeln für diese
14.)Artikelmenge und mache mit der nächsten Artikelmenge weiter.

Die Berechnung der häufigen Artikelmengen in Zeile 1 erfolgt mit dem weiter oben beschriebenen Algorithmus. Für jede Artikelmenge wird zunächst die Regel $X \rightarrow \{\}$ der Ergebnismenge aufgenommen, da diese immer eine Konfidenz von 100% hat (Zeile 3). Danach werden sukzessive Regeln mit größeren Konklusionen erzeugt (Zeile 6-10). Die Kandidatengenerierung erfolgt dabei analog zum Algorithmus zur Berechnung der häufigen Artikelmengen. Der Algorithmus terminiert, wenn alle häufigen Artikelmengen betrachtet wurden.
Die meiste Zeit für die Regelgenerierung wird durch den Algorithmus zur Berechnung der häufigen Artikelmengen in Anspruch genommen. Daher wurde der Algorithmus zur Regelgenerierung seit seiner Einführung auch kaum optimiert und die meiste Forschung konzentrierte sich auf Verbesserungen für den weiter oben beschriebenen Algorithmus zum Entdecken der häufigen Artikelmengen.

8.4.3 Optimierungen

Der **AprioriTid** Algorithmus versucht die Zeit für das Zählen des Support zu reduzieren. Dazu erzeugt er eine angepasste Datenbank, die mit $\overline{C_k}$ bezeichnet wird. $\overline{C_k}$ enthält alle Transaktionen der Datenbank, wobei die Transaktionen aber nur aus den häufigen k-Artikelmengen bestehen. Falls eine Transaktion keine häufigen k-Artikelmengen enthält wird sie aus $\overline{C_k}$ gestrichen. Der Vorteil ist, daß nach dem ersten Durchgang und dem Erzeugen von $\overline{C_1}$ keine Datenbankanfragen mehr nötig sind. Das Problem dieses Ansatzes besteht darin, daß er nur schnell ist, wenn $\overline{C_k}$ im Hauptspeicher gehalten werden kann. Bei den ersten Durchgängen mit kleinen Kandidaten ist die Menge $\overline{C_k}$ in der Regel so groß, daß sie nicht in den Hauptspeicher passt. Das Auslagern auf die Festplatte verlangsamt natürlich den Algorithmus. In späteren Durchgängen mit großen Kandidaten ist AprioriTid schneller als Apriori, da es nur noch wenige

Kandidaten gibt und somit die einzelnen Transaktionen in $\overline{C_k}$ sehr klein sind oder schon gestrichen wurden.
Der Algorithmus **AprioriHybrid** versucht die Vorteile beider Algorithmen zu kombinieren. Er benutzt in den frühen Durchgängen Apriori und schaltet dann später auf AprioriTid um, wenn erwartet wird, daß die Menge $\overline{C_k}$ in den Hauptspeicher passt. Hierzu wird eine Heuristik benutzt. Am Ende eines Durchgangs von Apriori kennt man den Support der Kandidaten in C_k. Da die Größe von $\overline{C_k}$ proportional zur Größe der Kandidatenmenge ist kann die Größe von $\overline{C_k}$ abgeschätzt werden. Wenn $\overline{C_k}$ klein genug ist, um in den Speicher zu passen und es in diesem Durchgang weniger Kandidaten als im vorherigen Durchgang gibt, wird auf AprioriTid umgeschaltet. AprioriHybrid hat fast immer eine bessere Laufzeit als Apriori, allerdings ist dies nicht allgemeingültig. Man betrachte nur den Fall, daß auf AprioriTid umgeschaltet wird, also die Menge $\overline{C_k}$ berechnet wird und im nächsten Durchgang keine neuen Kandidaten mehr gefunden werden. AprioriHybrid hat noch den Aufwand $\overline{C_k}$ zu berechnen, während Apriori sofort stoppen kann.

Betrachtet man die Kandidatengenerierung wird klar, daß es genau $\binom{F_1}{2}$ Kandidatenmengen der Größe 2 gib. Diese Menge ist oft zu groß für den Hauptspeicher. Aus der Praxis ist allerdings bekannt, daß viele dieser Kandidaten gar nicht in der Transaktionsdatenbank vorkommen. Bei der **dynamischen 2er-Kandidatengenerierung** werden die Kandidaten daher nicht aus F_1 sondern direkt aus der Datenbank erzeugt. Dazu wird, sobald F_1 feststeht, die Datenbank durchlaufen und alle nicht häufigen Artikel entfernt. In einem weiteren Durchlauf durch die Datenbank werden der Support der Kandidatenmengen der Größe 2 gezählt. Dabei wird ein Kandidat erst erzeugt, wenn er zum ersten mal auftritt. Kandidaten, die nicht vorkommen, werden damit nicht erzeugt. Dieses Vorgehen führt oft zu einer starken Reduzierung der Größe von C_2.

8.5 DFS-Algorithmen

Im Gegensatz zum Apriori Algorithmus, der einen Breitendurchlauf zur Erzeugung der häufigen Artikelmengen benutzt, verwenden die zwei nachfolgend beschriebenen Algorithmen einen Tiefendurchlauf. Man beachte, daß die Regelgenerierung hier nicht weiter betrachtet wird; die beiden Algorithmen erzeugen nur die häufigen Artikelmengen.

8.5.1 Eclat

Der Eclat Algorithmus verwendet das vertikale Datenbanklayout, d.h. die Datenbank besteht aus einer Menge von Artikeln mit ihren Covers. Die Berechnung des Supports einer Artikelmenge ist auf diese Weise einfach und schnell

möglich. Falls Y und Z Teilmengen von X sind und $Y \cup Z = X$ ist, bildet man die Schnittmenge $Y \cap Z$ und erhält das Cover von X. Der Support entspricht dann der Mächtigkeit des Covers von X. Diese Methode ist allerdings speicherplatzinitensiv. Die Cover aller Artikel repräsentieren die komplette Datenbank. Durch die Verwendung der DFS-Strategie kann der Speicherplatzbedarf stark reduziert werden, da immer nur ein kleiner Teil aller Covers im Speicher gehalten werden muss.
Gegeben sind die Transaktionsdatenbank TD und die Minimalhäufigkeit s_{min}. Zu Beginn werden alle Artikel aus der Datenbank entfernt, die nicht häufig sind. Eclat kennt zu jedem Zeitpunkt einen Präfix P und berechnet die Menge aller häufigen Artikelmengen, die diesen Präfix gemeinsam haben. Für den ersten Aufruf ist P={}. Der Algorithmus führt für jeden Artikel i in der Datenbank TD folgende Schritte durch:

- Nimm Artikel i in die Lösungsmenge auf, da TD nur häufige Artikel enthält.
- Erzeuge die i-projizierte Datenbank TD^i, die nur die Artikel $j > i$ enthält, die zusammen mit i eine häufige Artikelmenge ergeben. Auf diese Weise wird sichergestellt, daß die i-projizierte Datenbank wieder nur häufige Artikel enthält.
- Berechne die Menge aller häufigen Artikelmengen, die den Artikel i als Präfix enthalten. Dazu wird Eclat rekursiv mit der i-projizierten Datenbank TD^i und dem Prafix P = $\{i\}$ aufgerufen. Die Ergebnisse des rekursiven Aufrufs werden der Lösungsmenge hinzugefügt

Der Vorteil von Eclat ist, daß die Berechung des Supports bei vertikalem Datenbanklayout sehr schnell ist. DFS-Ansatz sorgt dafür, daß nur wenige Artikel(Cover) im Hauptspeicher gehalten werden müssen. Bei Tiefe d sind dies alle häufigen k-Artikelmengen mit demselben k-1 Präfix. Der BFS-Ansatz der von Apriori benutzt wird erfordert hingegen, daß alle k-Artikelmengen im Hauptspeicher sein müssen. Der Nachteil ist, daß Eclat die Support Monotonie nicht voll nutzten kann. Eclat generiert Kandidaten nur aus zwei seiner Teilmengen. Damit kann nicht getestet werden, ob eine seiner k-1 Teilmengen nicht häufig ist und der Kandidat muss auf jeden Fall erzeugt werden. Da die Transaktionsdatenbank im vertikalen Layout vorliegt, kann keine dynamische Kandidatengenerierung genutzt werden, da die Transaktionen nicht durchlaufen werden können.

8.5.2 FP-growth

FP-growth benutzt als Datenstruktur den sogenannten FP-Tree. Hierbei handelt es sich um eine Baumstruktur, mit der eine Datenbank in kompakter

Form dargestellt wird. Im Baum werden die Transaktionen gespeichert. Die Transaktionen werden nach Support geordnet und absteigend durchlaufen. Für jede Transaktion wird ein Pfad erzeugt, dessen Knoten mit den Artikeln der Transaktion benannt sind. Zusätzlich enthalten die Knoten einen Zähler. Wenn zwei Transaktionen den gleichen Präfix haben, teilen sie sich einen Knoten im Baum, was durch Erhöhen des Zählers dargestellt wird. Außerdem gibt es eine Item Header Tabelle, in der alle häufigen Artikel stehen. Für jeden Artikel gibt es eine lineare Liste, die alle Transaktionen durchläuft, die diesen Artikel enthalten. Der FP-Tree stellt eine Kombination aus vertikalem und horizontalem Datenbanklayout dar. Jede lineare Liste stellt eine komprimierte Form des Covers eines Artikels dar, während jeder an der Wurzel beginnende Zweig eine komprimierte Form einer Menge von Transaktionen ist. Alle Transaktionen, die den Artikel i enthalten, können leicht gefunden werden, indem man der linearen Liste folgt und von jedem erreichten Knoten den Weg zur Wurzel durchläuft. Dabei kann auch der Support aller Artikel j unter der Bedingung, daß i in der Transaktion vorkommt, gezählt werden.

TID	gekaufte Items	(geordnete) häufige Items
1	f,a,c,d,g,i,m,p	f,c,a,m,p
2	a,b,c,f,l,m,o	f,c,a,b,m
3	b,f,h,j,o	f,b
4	b,f,k,s,p	c,b,p
5	a,f,c,e,l,p,m,n	f,c,a,m,p

Abbildung 8.1: Beispiel einer Transaktionsdatenbank

In den Abbildungen 8.1 und 8.2 sind eine Transaktionsdatenbank und der zugehörige FP-Tree dargestellt. Man erkennt die komprimierte Darstellung der Transaktionsdatenbank im FP-Tree, die Item Header Tabelle und die lineare Liste, die für jeden Artikel alle Transaktionen durchläuft, die den Artikel enthalten.
Der FP-growth Algorithmus arbeitet ähnlich wie Eclat, allerdings wird als Da-

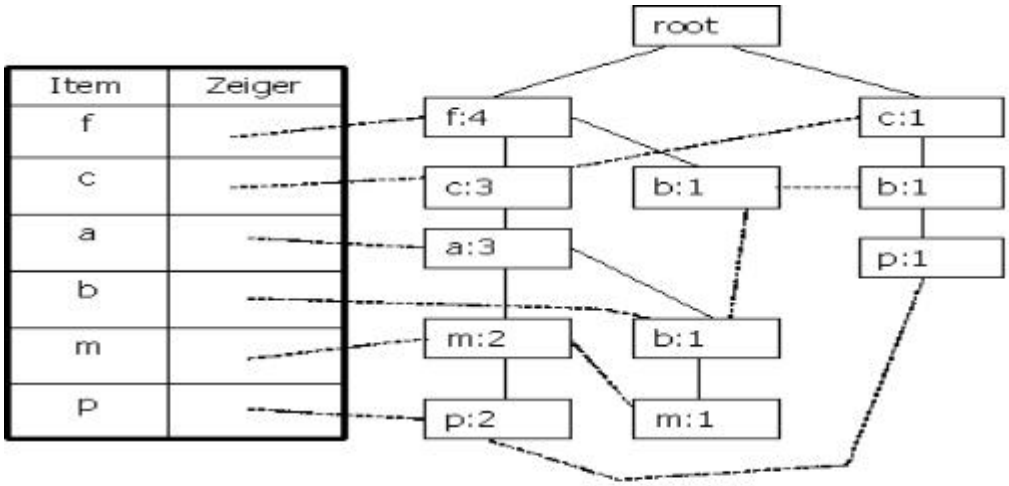


Abbildung 8.2: Beispiel eines FP-Tree

tenstruktur der FP-Tree verwendet. Für jeden Artikel i in der Header Tabelle führt der Algorithmus folgende Schritte aus:

- Nimm Artikel i in die Lösungsmenge auf.
- Erzeuge die i-projizierte Datenbank TD^i und den daraus resultierenden FP-Tree.
- Berechne die Menge aller häufigen Artikelmengen, die den Artikel i als Präfix enthalten. Dazu wird FP-growth rekursiv mit dem i-projizierten FP-Tree und dem Prafix P = $\{i\}$ aufgerufen. Die Ergebnisse des rekursiven Aufrufs werden der Lösungsmenge hinzugefügt.

Zur Erzeugung der i-projizierten Datenbank TD^i durchläuft man die lineare Liste für den Artikel i. Für jeden gefundene Knoten liefert der Weg zur Wurzel die Transaktionen, die i enthalten. Der Support des Artikels j in in den Transaktionen bestimmt dabei, wie oft dieser zu TD^i hinzugefügt wird. Es kann aber kein Artikel häufiger hinzugefügt werden als der Support von Artikel i ist. Artikel j, die nicht häufig sind, können wieder gestrichen werden, da sie mit Artikel i keine häufige Artikelmenge mehr ergeben können.
Betrachtet man noch einmal die Abbildung des FP-Trees kann der Prozeß leicht nachvollzogen werden. Der Artikel p steht in der Item Header Tabelle, ist somit häufig und wird in die Lösungsmenge aufgenommen. Dann wird, beginnend

in der Item Header Tabelle, die lineare Liste des Artikels p durchlaufen. Es werden zweimal die Transaktion (f,c,a,m,p) und einmal die Transaktion (c,b,p) gefunden. Zählt man die in den Transaktionen vorkommenden Artikel, ergibt sich mit der Minimalhäufigkeit von 3, daß nur Artikel c häufig ist. Also erfolgt der rekursive Aufruf mit Artikel p als Präfix und der i-projizierten Datenbank, die nur aus c besteht.
Der Vorteil von FP-growth liegt in der komprimierten Darstellung der Datenbank. Diese Komprimierung wird noch verbessert, indem die Artikel nach absteigendem Support sortiert werden. Da die Transaktionen im FP-Tree vorliegen, können die Kandidaten dynamisch generiert werden. Allerdings kann auch bei FP-growth die Support Monotonie nicht genutzt werden und die Akualisierung des FP-Trees verursacht Overhead. Der Kompressiongewinn ist oft nicht groß genug, um diesen Overhead auszugleichen, so daß Eclat häufig schneller ist als FP-growth.

8.6 Experimente

Name	#Items	#Transaktionen	min|T|	max|T|	avg|T|
T40I10D100K	942	100 000	4	77	39
Mushroom	119	8 142	23	23	23
BMS-Webview-1	497	59 602	1	267	2
Basket	13 103	41373	1	52	9

Name	s_min_{min}	$\|F_1\|$	$\|F\|$	$Max\{k \mid \|F_k\| > 0\}$
T40I10D100K	700	804	550 126	18
Mushroom	600	60	945 309	16
BMS-Webview-1	36	368	461 521	15
Basket	5	8051	285 758	11

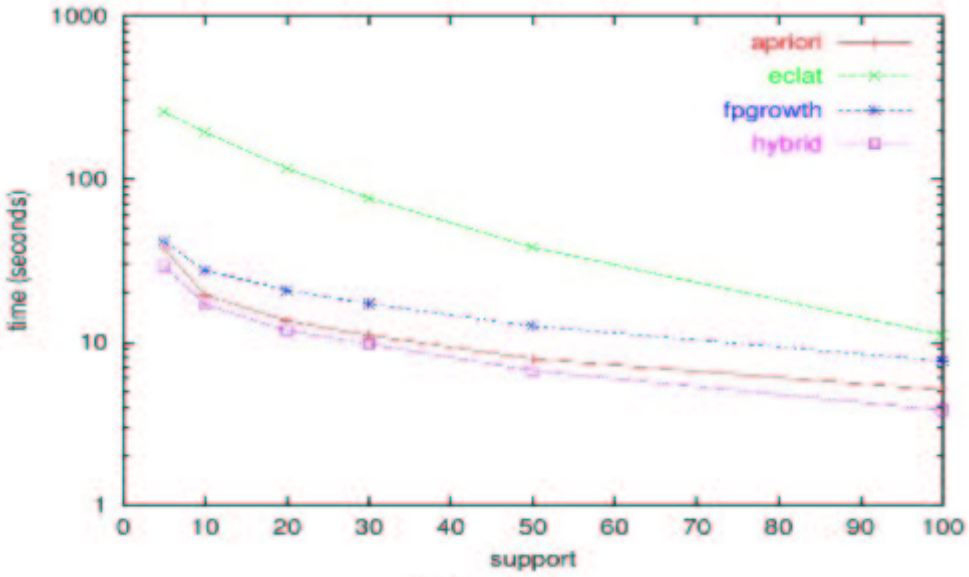
Abbildung 8.3: Beispieldatenbanken

Für die Experimente wurden vier verschiedene Datenbanken benutzt. T40I10D100K ist eine künstlich erzeugte Datenbank, Mushroom enthält Eigenschaften verschiedener Pilze, BMS-Webview sind über mehrere Monate erfasste Transaktionen eines Online-Händlers und Basket besteht aus den Transaktionen eines

belgischen Handelsunternehmens. Details zu den Datenbanken sind in Abbildung 8.3 dargestellt. Sie zeigt für die vier Beispieldatenbanken die Anzahl der Artikel und Transaktionen, die minimale, maximale und durchschnittliche Transaktionsgröße sowie den minimalen verwendeten Support, die Anzahl der häufigen Artikel, die Anzahl der häufigen Artikelmengen und die größten gefundenen häufigen Artikelmengen.
Testrechner war eine 400 MHz Sun Ultra Sparc 10 mit 512MB Hauptspeicher und Sun Solaris 8. Folgende Algorithmen wurden hiermit getestet:

- Apriori mit dynamischer 2er-Kandidatengenerierung
- Eclat
- FP-Growth
- Hybrid

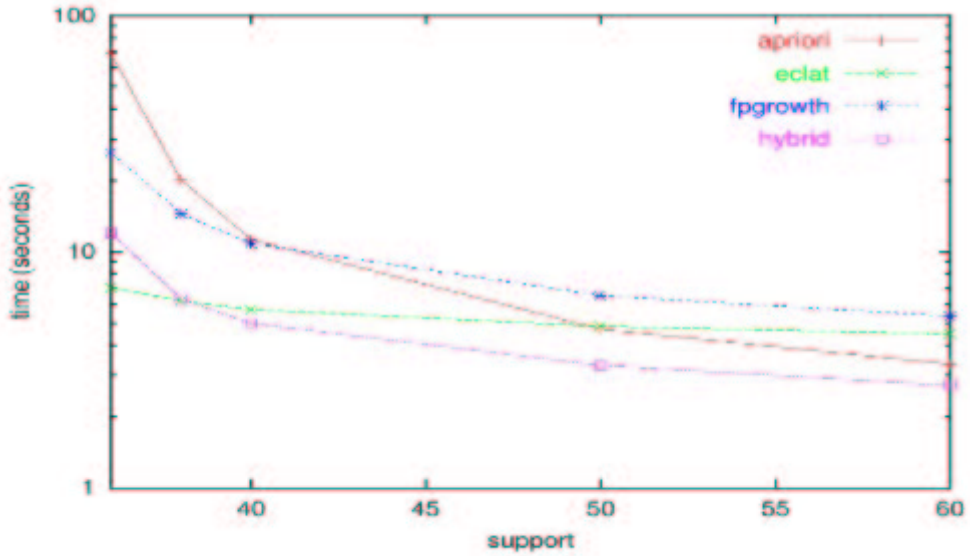
Hybrid verwendet zunächst Apriori und schaltet dann zu einem vom Nutzer bestimmten Zeitpunkt auf Eclat um. Implementiert wurde in C++. Die Ergebnisse sind in den Abbildungen 8.4, 8.5, 8.6 und 8.7 dargestellt.



(a) basket

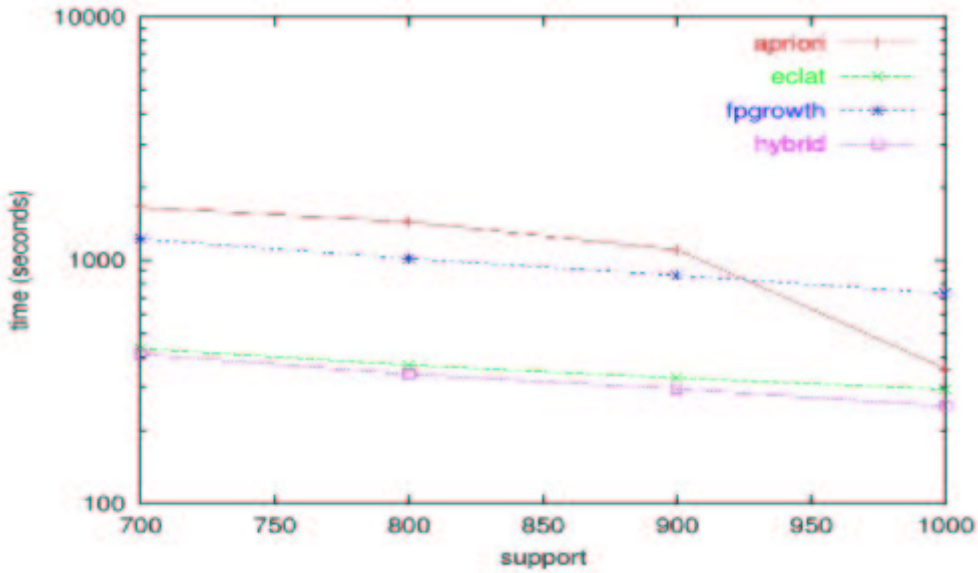
Abbildung 8.4: Ergebnis Basket

Für die Basket-Datenbank ist Eclat langsam. Das liegt daran, daß es sehr viele häufige Artikel gibt. Eclat benutzt als einziger Algorithmus keine dynamische



(b) BMS-Webview-1

Abbildung 8.5: Ergebnis BMS Webview



(c) T40I10D100K

Abbildung 8.6: Ergebnis T40I10D100K

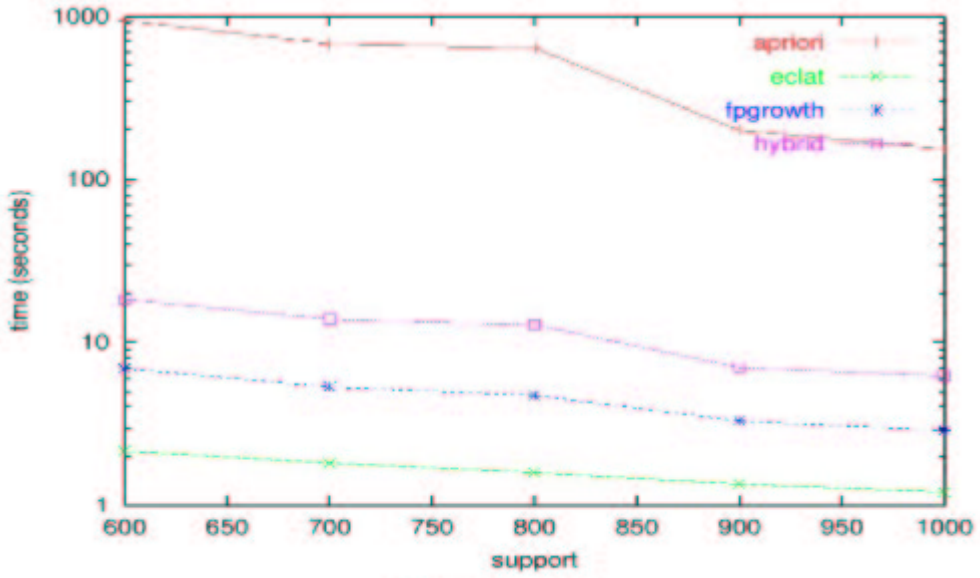


Abbildung 8.7: Ergebnis Mushroom

Kandidatengenerierung, so daß viele überflüssige 2er Kandidaten erzeugt werden. Hybrid funktioniert am besten, wenn nach der zweiten Iteration auf Eclat umgestellt wird. Apriori ist schneller als FP-growth, da die Datenbank dünn besetzt ist und die Aktualisierung des Supports für alle Kandidaten schneller geht als die Aktualisierung der Baumstruktur von FP-growth. Auch für die BMS-Webview-Datenbank arbeitet Hybrid am besten, wenn nach der zweiten Iteration gewechselt wird. Apriori funktioniert für niedrigen Support schlecht, da es einige sehr große Transaktionen mit über 100 häufigen Artikeln gibt. Apriori durchläuft für alle Kandidaten die Transaktionen und zählt den Support. Die Methode, um den Support zu zählen, ist bei Eclat (Schnittmenge) und FP-growth (FP-Tree) effektiver. Eclat ist schneller als FP-growth, da der FP-Tree zuviel Overhead erzeugt bzw. der Kompressionsgewinn nicht groß genug ist. Bei der künstlichen T40I10D100K-Datenbank arbeitet Apriori für niedrigen Support wieder schlecht, da viele Kandidaten generiert werden müssen. Hybrid funktioniert diesmal am besten, wenn nach der dritten Iteration umgeschaltet wird. Auch für die Mushroom-Datenbank ist Apriori schlecht. Jede Transaktion enthält genau 23 Artikel, wovon viele einen hohen Support haben. Das Zählen des Supports dauert damit wieder lange. Folglich funktioniert Hybrid am besten, wenn Apriori nicht benutzt wird. Hier wird die Performanz beim Umschalten nach der zweiten Iteration dargestellt.

8.7 Einordnung

Innerhalb des KDD-Prozesses läßt sich Apriori als konkretes Verfahren in die Phase des Data Minings einordnen. Im Gegensatz zu Verfahren wie Top-Down Induction of Decision Trees, Support Vector Machines, RDT oder Bagging und Boosting ist Apriori kein prädiktiv orientiertes Lernverfahren (Funktionslernen aus Beispielen), sondern ein deskriptives Lernverfahren. Bei prädiktiven Lernverfahren ist ein globales Modell erforderlich, daß für jede mögliche Instanz eine Vorhersage über den Funktionswert machen kann. Beim deskriptiven Lernen hingegen will man Teilbereiche des Instanzenraums identifizieren über die lokal interessante Aussagen gemacht werden können. Allerdings gibt es mit Apriori-C mittlerweile auch eine Implementierung, mit der Klassifikationen durchgeführt werden können. Das Lernen von Assoziationsregeln ist eine Instanz der Lernaufgabe Subgruppenentdeckung. Es ist aber technisch nicht sinnvoll die Entdeckung von Assoziationsregeln aus der Perspektive der Subgruppenentdeckung zu betrachten, da die besonderen Annahmen beim Apriori Algorithmus und die simple Qualitätsfunktion andere Suchverfahren erfordern. Eine offensichtliche Beziehung besteht zwischen Apriori und Apriori für zeitliche Daten, da letzteres eine Erweiterung des normalen Apriori Ansatzes darstellt. Beim Clustering soll eine gegebene Instanzenmenge aus einem Instanzenraum in Teilmengen aufgeteilt werden, so daß Objekte aus derselben Teilmengen möglichst ähnlich sind, während Objekte aus verschiedenen Teilmengen möglichst unähnlich sind. Dagegen werden bei Subgruppenentdeckung bzw. Apriori die Gruppen hinsichtlich ihrer Eigenschaften im Bezug auf ausgewählte Zielmerkmale identifiziert.

Kapitel 9

APRIORI für zeitliche Daten

Felix Jungermann

9.1 Einführung

Bei vielen Mechanismen des Maschinenlernens sowie des Dataminings werden ungeordnete Datenmengen behandelt. Es gibt jedoch viele Bereiche, in denen Sequenzen von „Vorkommnissen“ (sogenannten Events) betrachtet werden.

Als Beispiele seien hier Benutzeroberflächenaktionen und Kriminalstatistiken einer bestimmten Person genannt.

Abstrakt gesehen, kann man sagen, dass diese Daten Ereignis-Folgen sind, wobei jedes Ereignis (ab jetzt nur noch Event genannt) eine bestimmte Vorkommenszeitpunkt hat. Als Beispiel für eine Event-Folge kann man folgende Grafik anführen:

Abbildung 9.1: Krankenstatistik

Die Krankenstatistik zeigt über einen Zeitraum (hier vom 24. bis zum 44. Lebensjahr) die Erkrankungen einer Person und ihren Erkrankungszeitpunkt an.

Das Hauptproblem ist nun, in Sequenzen wie diesen häufige Vorfälle (soge-

nannte Episoden) zu finden. In diesem Fall zum Beispiel, dass nach einer bestimmten Erkrankung (D) auch eine andere (B) folgt. Episoden sind also Reihen von partiell geordneten Events.

Das Ziel für dieses Beispiel ist also, mögliche Krankheitsbilder und -folgen zu erkennen.

Es geht darum, aus einer Folge von Events und einer Klasse von Episoden die Episoden zu extrahieren, die möglichst oft in der Eventfolge vorkommen.

Zuerst werden Eventfolgen gesucht. Dann werden die Algorithmen zur Erkennung der häufigen Vorkommnisse (frequent episodes) beschrieben. Diese basieren darauf, zuerst kleine frequent episodes, und dann sukzessiv grössere zu finden.

Es werden die Algorithmen MINEPI und WINEPI vorgestellt.

Die Ausarbeitung stuetzt sich hauptsaechlich auf [51]

9.2 Eventfolgen und -episoden

Wie schon gesagt geht es hauptsächlich darum, Folgen von Events sowie wiederkehrende Kombinationen von Events zu finden, welche man häufige Episoden nennt.

9.2.1 Eventfolgen

Man sieht die Eingabe als eine Folge von Events an, wobei wie gesagt jeder Event einen Vorkommenszeitpunkt hat. Hat man einen Satz E von Event-Typen, so ist ein Event ein Paar (A,t), so dass $A \epsilon E$ ein Eventtyp und t eine Zahl ist, die die Erscheinungszeit des Events kennzeichnet.

Eine Eventfolge s auf E ist ein Tripel (s,T_s,T_e), und $s=\langle(A_1,t_1),(A_2,t_2),...,(A_n,t_n)\rangle$ $T_s$wird die Startzeit und $T_e$die Endzeit genannt. Für i=1,...n-1 gilt: $T_s \leq t_i <$ T_eMan betrachtet also nur alle Events, die zwischen Start- und Endzeit liegen.

Die Abbildung 9.1 präsentiert also graphisch die Eventfolge s=(s,24,44), wobei $s=\langle(C,24),(A,26),(D,29),(B,30),...,(B,44)\rangle$

In der Analyse dieser Folgen ist man nun daran interessiert, die häufigen Episoden einer Klasse von Episoden zu finden. Damit Episoden für uns interessant sind, müssen sie zeitlich gesehen nah beieinander liegen. Wie nah sie beieinander liegen sollen, bestimmt der Benutzer, indem er die Weite des Zeitfensters

angibt, in dem die Episode erscheinen muss.

Das Zeitfenster ist also ein Ausschnitt aus der gesamten Eventfolge. Die Eventfolge besteht also nach dem Definieren der Zeitfenster aus einer Sequenz sich teilweise überlappender Zeitfenster. Dann gibt der Benutzer an, in wie vielen Zeitfenstern eine Episode erscheinen muss, damit man sie als häufig ansehen kann.

Formal gesehen ist ein Zeitfenster auf einer Eventfolge $s=(s,T_s,T_e)$ eine eigene Eventfolge $w=(w,t_s,t_e)$, wobei $t_s<T_e$, $t_e>T_s$ist, und w aus den Paaren (A,t) aus s besteht, bei denen gilt: $t_s \leq t < t_e$. Die Zeitspanne te-ts ist die Weite des Fensters w. Mit W(s,win) wird der Satz aller Fenster w auf s bezeichnet, so dass die Weite der Fenster jeweils win beträgt. Das erste Fenster enthält nur den ersten Zeitpunkt, das letzte nur den letzten. In jedem Zeitpunkt wird ein neues Fenster geöffnet. Daher beträgt die Anzahl der Fenster $W(s,win)=T_e-T_s+win-1$

In dem Beispiel Abbildung 9.1 betrachten wir nun beispielhaft ein Fenster der Weite 5 von Zeitpunkt 24 bis 29. Das Fenster ist also: $(\langle(C,24),(A,26),(D,29)\rangle,24,29)$

9.2.2 Eventepisoden

Informell gesehen ist eine Episode eine partiell geordnete Sammlung von Events, die zusammen auftauchen.

Betrachtet man die Episoden in folgender Grafik, so ist a eine serielle Episode. Das bedeutet, dass a nur dann in einer Folge auftaucht, wenn die Events D und B in dieser Reihenfolge auftauchen.

Episode b ist eine parallele Episode. Es sind keine Beziehungen zwischen A und C bekannt.

Episode c ist ein Beispiel für eine nicht-serielle und nicht-parallele Episode. Diese erscheint, wenn A und C in der Folge auftauchen und dem Erscheinen von Event D vorangehen.

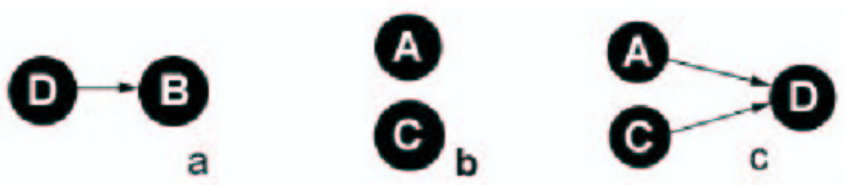


Abbildung 9.2: Episodentypen

Formal gesehen bezeichnet man Episoden als Tripel (V,≤,g) wobei V ein Satz aus Knoten, ≤ eine partielle Ordnung über V und g:V→E eine Abbildung jedes Knotens auf einen Eventtyp ist.

Eine Episode d ist parallel, wenn die partielle Ordnung ≤ trivial ist, also x≠y für alle x,y ϵ V , so dass x≠y. Episode d ist seriell, wenn die Relation ≤ eine totale Ordnung ist, also x≤y oder y≤x für alle x,y ϵ V. d ist injektiv, wenn die Abbildung g injektiv ist, also kein Eventtyp zweimal in der Episode auftaucht.

Wir betrachten als Beispiel Episode a aus Abbildung 9.2. V enthalte zwei Knoten: zum Beispiel x und y. Die Abbildung g bildet diese auf die Eventtypen in der Grafik ab. g(x)=D und g(y)=B. Ein Event des Typs D scheint vor einem Event des Typen B zu erscheinen. Also taucht x vor y auf, und wir erhalten x≤y. Die Episode ist zudem injektiv, da sie keine doppelten Eventtypen beinhaltet. In einem Fenster, in dem a erscheint, können jedoch mehrere Events der Typen D und B auftauchen.

Als nächstes definieren wir Subepisoden. Diese Relation werden wir oft in den folgenden Algorithmen benutzen, um alle häufigen Episoden zu erfassen. Eine Episode e=(V', ≤',g') ist eine Subepisode von f=(V,≤,g), notiert als e≤f, wenn es eine injektive Abbildung h:V' für alle v ϵ V'→V gibt, so dass g'(v)=g(h(v)) für alle v ϵ V' und für alle v,w ϵ V' mit v ≤' w und h(v) ≤ h(w) gilt. Eine Episode f ist Superepisode von e wenn e ≤ f. In Grafik 2 sieht man, dass b ≤ c, also b eine Subepisode von c ist.

Was bedeutet es nun, wenn eine Episode in einer Folge auftaucht? Eine Episode a = (V,≤,g) erscheint in einer Eventfolge s=($\langle(A_1,t_1),(A_2,t_2),...,(A_n,t_n)\rangle$,$T_s$,$T_e$), sofern e seine injektive Abbildung h:V→1,...,n von Knoten zu Events gibt, so dass g(x)=$A_{h(x)}$für alle x ϵ V gilt, und für alle x,y ϵ V mit x ≠ y und x ≤ y $t_{h(x)}<t_{h(y)}$gilt.

Das Fenster (w,24,29) für Abbildung 9.1 enthält die Events C, A und D. Die Episoden b und c der Abbildung 9.2 tauchen in dem Fenster auf, a hingegen nicht.

Nun definieren wir die Frequenz einer Episode. Diese ist die Anzahl der Fenster, in denen die Episode auftaucht. Die Frequenz einer Episode a in einer Eventfolge ist:

$$fr(a, s, win) = |\{w \epsilon W(s, win)| \text{ a erscheint in w}\}|/|W(s, win)|,$$

wobei s die Eventfolge und win die Fensterweite ist. $|W(s, win)|$ ist hierbei der Satz aller Fenster w auf s, so dass die Weiten der Fenster w allesamt win sind.

Man bezeichnet a als häufig (frequent), wenn fr(a,s,win) $\geq$ min_fr. min_fr ist der sogenannte frequency threshold, die Grenze, die man festlegt, um häufige Episoden zu erkennen. Die Sammlung häufiger Episoden wird mit F(s, win, min_fr) bezeichnet.

Hat man erstmal die häufigen Episoden erkannt, so ist es möglich, mit ihrer Hilfe Regeln zu erstellen, die Verknüpfungen zwischen den einzelnen Events herstellen. Wenn man weiss, dass Episode b aus Grafik 8.2 in 4,2% der Fenster und ihre Oberepisode c in 4% der Fenster auftaucht, dann kann man annehmen, dass in einem Fenster mit A und C zu 95%iger (4% / 4,2%) Wahrscheinlichkeit auch D (in dem gleichen Fenster) folgt.

Wie Regeln aus den Frequenzen von Episoden erstellt werden können, zeigt folgender Algorithmus:

Eingabe: Ein Satz E von Eventtypen, eine Eventsequenz s über E, ein Satz ϵ von Episoden, eine Fensterweite win, ein Frequenzschwellenwert min_fr sowie ein Konfidenzschwellenwert min_conf.

Ausgabe: Die Episodenregeln, die auf s existieren, mit Rücksicht auf win, min_fr und min_conf.

Methode:

1. /* Finde häufige Episoden (Algorithmus 2): */

2. berechne $F(s, win, min\_fr)$;

3. /* Regelerzeugung: */

4. for all $\alpha \epsilon F(s, win, min\_fr)$ do

5. for all $\beta < \alpha$ do

6. if $fr(\alpha)/fr(\beta) \geq min_$conf then

7. gebe die Regel $\beta \rightarrow \alpha$ und die Konfidenz $fr(\alpha)/fr(\beta)$ aus.

9.3 Algorithmen

Als erstes werden wir eine Spezifikation des Algorithmus liefern. Danach werden die exakten Methoden sowie seine Unterprogramme geliefert. Die folgenden Methoden werden als WINEPI Algorithmus bezeichnet.

9.3.1 Hauptalgorithmus

Eingabe: Ein Satz E von Eventtypen, eine Eventsequenz s über E, ein Satz ϵ von Episoden, eine Fensterweite win und ein Frequenzschwellenwert min_fr.

Ausgabe: Die Sammlung F(s, win, min_fr) häufiger Episoden.

Methode:

1. berechne $C_1 := \alpha\epsilon\varepsilon||\alpha| = 1$;

2. $l := 1$;

3. while $C_l \neq \emptyset$ do

4. /* Datenbankdurchlauf (Algorithmus 4 und 5): */

5. berechne $F_l := \alpha\epsilon\varepsilon|fr(\alpha, s, win) \geq min\_fr$;

6. $l := l + 1$;

7. /* Kandidatengenerierung (Algorithmus 3): */

8. berechne $C_l :=\alpha\epsilon\varepsilon||\alpha| = l$ und für alle $\beta\epsilon\varepsilon$, so dass $\beta < \alpha$ und $|\beta| < l$ ist, haben wir $\beta\epsilon F_{|\beta|}$;

9. for all l do output F_l;

Algorithmus 2 erstellt die Sammlung der häufigen Episoden von der Klasse E der Episoden.

Der Algorithmus macht eine levelweise Suche in dem Episodengitter, das von der Subepisodenrelation gespannt wird. Die Suche startet bei den einfachsten Episoden, also denen, die nur einen Event haben. In jeder Runde erstellt der Algo zuerst eine Menge von möglichen Episoden, und dann überprüft er ihre Frequenz in der Eventfolgen-Datenbank.

Lemma 1: Wenn eine Episode a häufig (frequent) in einer Eventfolge ist, dann sind auch alle Subepisoden b $\leq$ a häufig.

9.3.2 Erstellung der Episodenkandidaten

Algorithmus 3 erstellt Kandidaten für parallele Episoden.

Eingabe: Ein sortiertes Array Fıhäufiger paralleler Episoden der Grösse l.

Ausgabe: Ein sortiertes Array der Kandidaten paralleler Episoden der Grösse l+1.

Methode:

1. $C_{l+1} := \emptyset$;

2. $k := 0$;

3. if $l = 1$ then for $h := 1$ to $|F_l|$ do F_l.block_start$[h] := 1$;

4. for $i := 1$ to $|F_l|$ do

5. current_block_start:= $k + 1$;

6. for $(j := i; F_l$.block_start$[j] = F_l$.block_start$[i]; j := j + 1)$ do

7. /* $F_l[i]$ und $F_l[j]$ haben die ersten l-1 Eventtypen gemeinsam, erstelle einen potentiellen Kandidaten α durch ihre Kombination: */

8. for $x := 1$ to l do $\alpha[x] := F_l[i][x]$;

9. $\alpha[l + 1] := F_l[j][l]$;

10. /* Erstelle und teste Subepisoden β, die nicht $\alpha[y]$ enthalten: */

11. for $y := 1$ to $l - 1$ do

12. for $x := 1$ to $y - 1$ do $\beta[x] := \alpha[x]$;

13. for $x := y$ to l do $\beta[x] := \alpha[x + 1]$;

14. falls β nicht in F_l ist, so setze mit dem nächsten j bei Zeile 6 fort;

15. /* Alle Subepisoden sind F_l , speichere α als Kandidaten: */

16. $k := k + 1$;

17. $C_{l+1}[k] := \alpha$;

18. C_{l+1}.block_start$[k]$:=current_block_start;

19. gebe C_{l+1} aus;

Sofern die Episoden und Episodensammlungen sortiert sind, sind alle Episoden, die die ersten gleichen Eventtypen beinhalten, in der gleichen Episoden-

sammlung. Wenn also Episoden F_l[i] und F_l[j] der Grösse l die ersten l-1 Events beinhalten, dann gilt für alle k mit i≤k≤j, dass F_l[k] auch die gleichen Events innehat.

Eine maximale Folge von fortlaufenden Episoden der Grösse l, die die ersten l-1 Events beinhaltet, wird Block genannt.

Algorithmus 3 findet auch serielle Kandidaten, sofern man Zeile 6 durch folgende Zeile ersetzt:

6. for(j:= F_l.block_start[i]; F_l.block_start[j]=F_l.block_start[i];j:=j+1) do

Dieser Algorithmus (3) benötigt O ($l^2|Fl|^2 log|Fl|$)!

9.3.3 Erkennung von Episoden in Folgen

Nun geht es darum, die Datenbank zu durchlaufen. Für zwei Fenster w=(w,t_s, t_s+win) und w'=(w', t_s+1, t_s+win+1) sind die Folgen w und w' von Events sehr ähnlich. Diese Ähnlichkeit machen wir uns zu Nutze: nachdem Episoden in w erkannt wurden, macht man inkrementelle Updates in der Datenstruktur, um ein Verschieben des Fensters zu erreichen, damit man w' erhält.

Parallele Episoden Algorithmus 4 erkennt Kandidaten paralleler Episoden in Eventfolgen.

Eingabe: Eine Sammlung C paralleler Episoden, eine Eventfolge s=(s,T_s,T_e), eine Fensterweite win und ein frequency threshold min_fr. Ausgabe: Die Episoden aus C, die häufig in s sind.

Methode:

1. /* Initialisierung: */

2. for each α in C do

3. for each A in α do

4. $A.count := 0$;

5. for $i := 1$ to $|\alpha|$ do $contains(A, i) := \emptyset$;

6. for each α in C do

7. for each A in α do

8. $a :=$ Anzahl der Events des Typs A in α;

9. $contains(A, a) := contains(A, a) \cup \alpha$;

10. $\alpha.event\_count := 0$;

11. $\alpha.freq\_count := 0$;

12./* Erkennung: */

13.for $start := T_s - win + 1$ to T_e do

14. /* Neue Events ins Fenster holen: */

15. for all events (A, t) in s such that $t = start + win - 1$ do

16. $A.count := A.count + 1$;

17. for each $\alpha \epsilon contains(A, A.count)$ do

18. $\alpha.event\_count := \alpha.event\_count + A.count$;

19. if $\alpha.event\_count = |\alpha|$ then $\alpha.inwindow := start$;

20. /* alte Events aus dem Fenster fallen lassen */

21. for all events (A, t) in s such that $t = start - 1$ do

22. for each $\alpha \epsilon contains(A, A.count)$ do

23. if $\alpha.freq\_count = |\alpha|$ then

24. $\alpha.freq\_count := \alpha.freq\_count - \alpha.inwindow + start$;

25. $\alpha.event\_count := \alpha.event\_count - A.count$;

26. $A.count := A.count - 1$;

27./* Ausgabe */

28. for all episodes α in C do

29. if $\alpha.freq\_count/(T_e - T_s + win - 1) \geq min\_fr$ then output α;

Dieser Algorithmus hat eine Laufzeit von O($(n + l^2)|C|$). Die Initialisierung benötigt O($|C|l^2$). Das Fenster wird O(n) mal geshiftet, und die Kosten der

Erkennungsphase betragen O(n$|C|$).

Die Laufzeit zur Erkennung injektiver paralleler Episoden (ohne Initialisierung) beträgt O(($|C|l + n$) * (n/win)).

Für jeden Kandidat einer parallelen Episode α benutzen wir einen Zähler α.event_count, welcher anzeigt, wie viele Events von α im Fenster vorhanden sind. Falls α.event_count den Wert $|\alpha|$ annimmt, dann wird die Startzeit des Fensters in α.inwindow gespeichert. Wenn α.event_count wieder abnimmt, also α nicht mehr vollständig im Fenster vorhanden ist, erhöhen wir das Feld α.freq_count um die Anzahl von Fenstern, in denen α noch vollständig enthalten ist. Zum Schluss enthält α.freq_count die Anzahl aller Fenster, in denen α vorkommt.

Um auf Kandidaten schneller zuzugreifen, werden sie mit der Nummer des Events, den sie enthalten, indiziert. Episoden, die genau a Events vom Typ A enthalten, werden in die Liste contains(A,a) gespeichert.

Serielle Episoden

Serielle Kandidaten von Episoden werden mit Status Automaten erkannt. Diese akzeptieren nur den Episodenkandidaten und blockieren ansonsten. Für jede serielle Episode α gibt es einen Automaten, und es kann weiterhin mehrere Instanzen jedes Automaten zur selben Zeit geben.

Algorithmus 5 benutzt diese Idee.

Falls das erste Event einer Episode a in ein Fenster kommt, wird eine neue Instanz des Automaten für α erzeugt. Falls das Event das Fenster verlässt, wird der Automat gelöscht. Erreicht ein Automat für α den akzeptierenden Zustand, also ist α vollkommen im Fenster enthalten, wird der Startzeitpunkt des Fensters in α.inwindow gespeichert. Wird der Automat im akzeptierenden Zustand gelöscht, und ist kein anderer Automat für α im akzeptierenden Zustand, erhöhen wir das Feld α.freq_count um die Anzahl der Fenster, in denen α vollständig enthalten ist.

Die Automaten sind wie folgt organisiert: Für jeden Eventtypen A ϵ E ist der Automat, der A akzeptiert mit einer Liste (waits(A)) verbunden. Die Liste enthält Einträge der Form (α,x), was bedeutet, dass α auf den xten Event wartet.

Eingabe: Eine Sammlung C serieller Episoden, eine Eventfolge s=(s,T_s,T_e), eine Fensterweite win und ein frequency threshold min_fr.

Ausgabe: Die Episoden aus C, die häufig in s sind.

Methode:

1. /* Initialisierung: */

2. for each α in C do

3. for $i := 1$ to $|\alpha|$ do

4. $\alpha.initialized[i] := 0$;

5. $waits(\alpha[i]) := \emptyset$;

6. for each $\alpha \epsilon C$ do

7. $waits(\alpha[1]) := waits(\alpha[1]) \cup (\alpha, 1)$;

8. $\alpha.freq\_count := 0$;

9. for $t := T_s - win$ to $T_s - 1$ do $beginsat(t) := \emptyset$;

10./* Erkennung: */

11.for $start := T_s - win + 1$ to T_e do

12. /* hole neue Events ins Fenster */

13. $beginsat(start + win - 1) := \emptyset$;

14. $transitions := \emptyset$;

15. for all events (A, t) in s such that $t = start + win - 1$ do

16. for all $(\alpha, j) \epsilon waits(A)$ do

17. if $j = |\alpha|$ and $\alpha.initialized[j] = 0$ then $\alpha.inwindow := start$;

18. if $j = 1$ then

19. $transitions := transitions \cup (\alpha, 1, start + win - 1)$;

20. else

21. $transitions := transitions \cup (\alpha, j, \alpha.initialized[j - 1])$;

22. $beginsat(\alpha.initialized[j-1]) := beginsat(\alpha.initialized[j-1])$
$(\alpha, j-1)$;

23. $\alpha.initialized[j-1] := 0$;

24. $waits(A) := waits(A)$
(α, j);

25. for all $(\alpha, j, t) \epsilon transitions$ do

26. $\alpha.initialized[j] := t$;

27. $beginsat(t) := beginsat(t) \cup (\alpha, j)$;

28. if $j < |\alpha|$ then $waits(\alpha[j+1]) := waits(\alpha[j+1]) \cup (\alpha, j+1)$;

29. /* alte Events aus dem Fenster herausfallen lassen */

30. for all $(\alpha, l) \epsilon beginsat(start - 1)$ do

31. if $l = |\alpha|$ then $\alpha.freq\_count := \alpha.freq\_count - \alpha.inwindow + start$;

32. else $waits(\alpha[l+1]) := waits(\alpha[l+1])$
$(\alpha, l+1)$;

33. $\alpha.initialized[l] := 0$;

34./* Ausgabe */

35.for all episodes α in C do

36. if $\alpha.freq\_count/(T_e - T_s + win - 1) \geq min\_fr$ then output α;

Die Laufzeit dieses Algorithmus beträgt O(n $|C|$ l). Die Initialisierung benötigt O($|C|$ l +win). In der Erkennungsphase hat man wieder O(n) shifts. In einem Shift hängen die Bemühungen für eine Episode a von der Anzahl der Automaten ab. Für jede Episode gibt es höchstens l Automaten. Die worst-case Zeit beträgt also O($|C|l + win + n|C|$l) = O(n$|C|$l).

Die Laufzeit zur Erkennung injektiver serieller Episoden beträgt O(n$|C|$).

9.4 Alternativer Ansatz: minimale Vorkommen

9.4.1 Umriss des Ansatzes

Anstatt andauernd die Fenster zu betrachten und darauf zu warten, ob eine Episode auftaucht oder nicht, kann man auch das genaue Vorkommen der Episoden und ihrer Beziehungen untereinander beobachten. Der Vorteil dieses Ansatzes ist, dass das Beachten der exakten Vorkommen von Episoden uns erlaubt, Regeln über eine Weite von zwei oder mehr Fenstern zu finden. Regeln wie zum Beispiel: „Wenn A und B innerhalb von 15 Sekunden erscheinen, folgt C innerhalb von 30 Sekunden!“

Der Ansatz beruht auf den minimalen Vorkommen von Episoden.

Neben der Aufstellung von Regeln benötigen wir eine neue Methode : MINEPI, um die Episoden in der Eingabefolge zu erkennen.

Für jede häufige Episode speichern wir Informationen über die Orte ihrer minimalen Vorkommen. In der Erkennungsphase errechnen wir die Orte der minimalen Vorkommen einer Kandidatenepisode a aus der temporalen Verknüpfung der minimalen Vorkommen aus zwei Subepisoden von a.

Identifiziert werden minimale Vorkommen und ihre Zeitintervalle wie folgt: In einer gegebenen Episode α und einer Eventfolge s ist das Intervall $[t_s,t_e)$ ein minimales Vorkommen auf α in s, wenn 1.) α in dem Fenster w = (w,t_s,t_e) auf s auftaucht und 2.) α nicht in irgendeinem Unterfenster von w noch in einem Fenster w' = (w', t_s', t_e') auf s, so dass $t_s \leq t_s'$, $t_e' \leq t_e$und width(w')<width(w) ist, erscheint.

Der Satz minimaler Vorkommen einer Episode α in einer Eventfolge wird notiert als mo(α): mo(α) = $[t_s,t_e)|[t_s,t_e)$ ist ein minimales Vorkommen von α

Wenn man sich an die Eventfolge s aus Abbildung 9.1 sowie die Episoden aus Abbildung 9.2 erinnert, so sieht man, dass die parallele Episode b der Eventtypen A und C drei minimal Vorkommen in s hat: mo(b) = [24,27), [34,39), [38,40). Die partiell geordnete Episode c hat die folgenden zwei minimalen Vorkommen: [24,30), [38,43).

Eine Episodenregel ist ein Ausdruck b[win_1] $\rightarrow$ a[win_2], wobei b und a Episoden sind, für die gilt: b $\leq$ a. Ausserdem sind win_1und win_2ganze Zahlen.

Formal kann dies wie folgt ausgedrückt werden: win_1und b sind gegeben, mo_{win1}(b)=$[t_s,t_e)$ ϵ mo(b) | t_e- $t_s \leq$ win_1 Weiterhin gegeben sind: a und ein Intervall $[u_s,u_e)$, definiere occ(a,$[u_c,u_e)$)= true, wenn NUR ein minimales Vorkommen

[u_s',u_e') ϵmo(a) existiert, so dass u_s=u_s' und u_e'=u_e. Die Richtigkeit einer Episodenregel b[win1] $\rightarrow$ a[win2] ist jetzt:

$$(|[ts, te)\epsilon mowin1(b)|occ(a, [ts, ts + win2))|)/(|mowin1(b))$$

Die informelle Beschreibung der Regel ist, dass Episode b eine minimale Erscheinung im Intervall [t_s,t_e) mit t_e- $t_s$$\leq$ win$_1$hat, wenn Episode a im Intervall [t_s,t_e') für ein t_e' erscheint, so dass t_e'-$t_s$$\leq$ win$_2$.

Um beim eben angesprochenen Beispiel zu bleiben: Für die Regel b[3] $\rightarrow$ c[5] haben wir $|[24, 27), [38, 40)|$ im Nenner und $|[38, 43)|$ im Zähler, so dass die Richtigkeit $\frac{1}{2}$ ist. Bei b[3] $\rightarrow$ c[6] wäre die Richtigkeit 1.

Der Begriff der Frequenz ist bei minimalen Vorkommen nicht sehr nützlich, da

1.) es keine feste Fenstergrösse gibt und

2.) ein Fenster mehrere minimale Vorkommen haben könnte.

Anstatt von Frequenz benutzt man hier das Konzept der Unterstützung (support), die Anzahl der minimalen Vorkommen einer Episode. Der support einer Episode α in einer gegebenen Eventfolge s ist also $|mo(\alpha)|$. Auch hier benutzen wir einen threshold-Wert min_sup. Eine Episode nennt man also häufig (frequent), wenn $|mo(\alpha)| \geq$ min_sup.

9.4.2 Entdeckung minimaler Vorkommen von Episoden

Nun wird informell beschrieben, wie Algorithmen die minimalen Vorkommen häufiger serieller und paralleler Episoden entdecken.

Es gilt weiterhin, dass Subepisoden häufiger Episoden auch häufig sind.

Folgende Aussagen lassen sich über minimale Vorkommen einer Episode treffen, deren Subepisoden auch minimale Vorkommen hat.

Aussage 1: a ist eine Episode und b$\leq$a ist ihre Subepisode. Wenn [t_s,t_e) ϵ mo(a), dann erscheint b in [t_s,t_e), also gibt es ein Intervall [u_s,u_e) ϵ mo(b), so dass $t_s$$\leq$$u_s$$\leq$$u_e$$\leq$$t_e$.

Aussage 2: Sei a eine serielle Episode der Grösse k, und sei [t_s,t_e) ϵ mo(a). Dann gibt es Subepisoden a_1 und a_2 von a der Grösse k-1, so dass [t_s,t_e^1) ϵ mo(a_1) für einige t_e^1<t_e und [ts^2te) ϵ mo(a_2) für einige $ts^2 > ts$.

Aussage 3: Wenn a eine parallele Episode der Grösse k und [t_s ,t_e) ϵ mo(a) ist, dann gibt es Subepisoden a_1 und a_2 von a der Grösse k-1, so dass [t_s ^{1},t_e ^{1})ϵ

mo(a_1) und [$t_s{}^2$,$t_e{}^2$) ϵ mo(a_2) für einige $t_s{}^1$,$t_e{}^1$,$t_s{}^2$,$t_e{}^2$ ϵ [t_s, t_e] gilt. Weiterhin gilt: $t_s = \min(t_s{}^1, t_s{}^2)$ und $t_e = \max(t_e{}^1, t_e{}^2)$.

Die minimalen Vorkommen der Kandidaten Episode a werden in folgender Weise lokalisiert. Bei der ersten Iteration des Hauptalgorithmus wird mo(a) aus der Eingabe aller Episoden a der Grösse 1 berechnet. Im Rest der Durchläufe werden die minimalen Vorkommen eines Kandidaten a wie folgt lokalisiert: Zuerst werden zwei passende Subepisoden a_1 und a_2 von a gewählt. Dann werden die minimalen Vorkommen der beiden Subepisoden mit Hilfe der Aussagen 2 und 3 verbunden.

Zur Erläuterung: für serielle Episoden werden die zwei Subepisoden so gewählt, dass die erste alle Events bis auf den letzten und die zweite Episode alle Events bis auf den ersten enthält. Die minimalen Vorkommen von a findet man dann so:

mo(a)=[t_s,u_e) | es gibt [t_s,t_e) ϵ mo(a_1) und [u_s,u_e) ϵ mo(a_2), so dass $t_s < u_s$, $t_e < u_e$ und [t_s,u_e) minimal sind.

Bei parallelen Episoden enthalten die beiden Subepisoden alle Events bis auf einen, und die Subepisoden müssen verschieden sein.

Die minimalen Vorkommen einer Kandidatenepisode a können in einem linearen Durchlauf über die minimalen Vorkommen der Subepisoden a1 und a2 gefunden werden. Dafür benötigt man (für einen Kandidaten): $O(|mo(a1)| + |mo(a2)| + |mo(a)|)$, was O(n) ist, wobei n die Länge der Eventfolge ist.

9.4.3 Wahrheitsgehalt der gefundenen Regeln

Wir zeigen nun, wie die Informationen über minimale Vorkommen von häufigen Episoden genutzt werden können, um den Wahrheitsgehalt einer Episodenregel zu erfahren, ohne ein weiteres mal die Daten zu betrachten.

Wie schon erwähnt ist eine Episodenregel ein Ausdruck b[win_1] $\rightarrow$ a[win_2], wobei b und a Episoden sind, für die gilt: b $\leq$ a und win_1 und win_2 sind ganze Zahlen.

Damit die Regeln allgemeingültig (frequent) sind, muss auch die Episode a häufig sein. Der Wahrheitsgehalt einer gefundenen Regel kann in einem Durchlauf durch die Strukturen mo(b) und mo(a) entwickelt werden. Für jedes [t_s,t_e) ϵ mo(b) mit $t_e - t_s \leq win_1$, markiere das minimale Vorkommen [u_s,u_e)von a, so dass $t_s \leq u_s$ und [u_s,u_e) das erste Intervall in mo(a) mit diesen Eigenschaften ist. Dann überprüft man ob $u_e - t_s \leq win_2$.

9.5 Experimente

9.5.1 Performanz

Die zu behandelnde Datenbank ist eine Folge von 73679 Alarmmeldungen, die über einen Zeitraum von 7 Wochen erhoben wurden. Es gibt 287 verschieden Alarmtypen mit jeweils unterschiedlichen Frequenzen und Verteilung.

Als erstes wird WINEPI, wie er in Punkt 3 beschrieben wurde, betrachtet. Hierbei werden die Resultate zweier völlig verschiedener Komplexitätsfälle betrachtet: serielle Episoden sowie injektive parallele Episoden.

Die nächsten beiden Grafiken zeigen die Performanzstatistik beim Finden häufiger Episoden in der Alarmdatenbank mit unterschiedlichen frequency thresholds.

Frequenzschwellenwert	Kandidaten	Häufige Episoden	Iterationen	Zeit (in s)
0,001	4528	359	45	680
0,002	2222	151	44	644
0,005	800	48	10	147
0,010	463	22	7	110
0,020	338	10	4	62
0,050	288	1	2	22
0,100	287	0	1	16

Tabelle 9.1: Serielle Episoden

Die Anzahl der häufigen Episoden nimmt dabei rapide ab, sobald der threshold-Wert zunimmt.

Frequenzschwellenwert	Kandidaten	Häufige Episoden	Iterationen	Zeit (in s)
0,001	2122	185	5	49
0,002	1193	93	4	48
0,005	520	32	4	34
0,010	366	17	4	34
0,020	308	9	3	19
0,050	287	1	2	15
0,100	287	0	1	14

Tabelle 9.2: Injektiv parallele Episoden

Abbildung 9.3 zeigt das Verhalten bei unterschiedlicher Fensterweite.

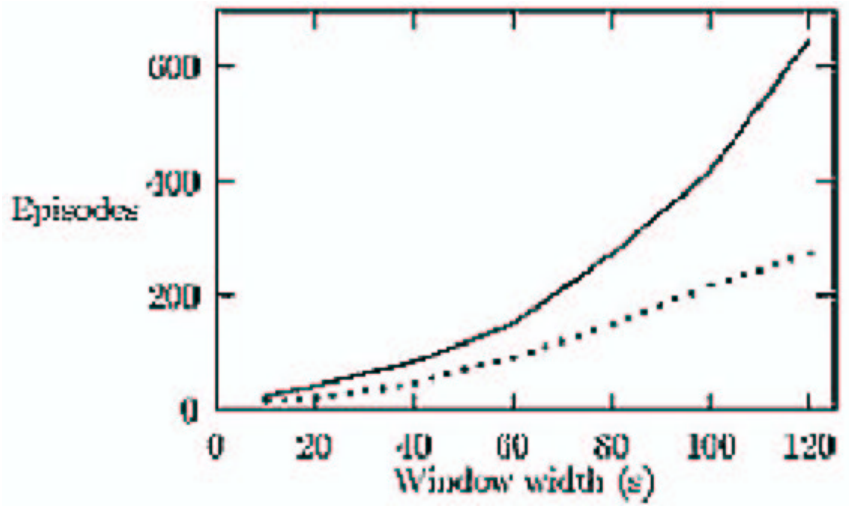


Abbildung 9.3: Verhalten bei unterschiedlicher Fensterweite

Abbildung 9.4 zeigt die Anzahl der seriellen sowie injektiv parallelen Episoden, die durch unsere Methode gefunden werden.

Abbildung 9.5 zeigt die Rechenzeit, die bei sich erhöhendem frequency threshold-Wert immer kleiner wird.

Die Rechenzeit ist für parallele Episoden viel kleiner als für serielle mit dem gleichen threshold-Wert. Dafür gibt es zwei Gründe. Erstens sind die parallelen Episoden kürzer als die seriellen, und zweitens braucht man weniger Datenbankdurchläufe.

9.5.2 Qualität der Kandidatengenerierung

Was passiert bei den ersten Iterationen? Statistiken der ersten 10 Iterationen eines Durchlaufs mit einem frequency threshold von 0.001 und einer Fensterweite von 60 s sieht man in Abbildung 9.3. Erst nach den ersten drei Iterationen werden die Kandidaten-Generationen effizient. Vorher gibt es viel zu viele Kandidaten, von denen weniger als 20% häufig (frequent) sind. Die Anzahl der Iterationen beträgt insgesamt 45, jedoch behandeln die letzten 35 jeweils nur ein bis drei Kandidaten, so dass man ruhig ein paar der späteren Iterationen kombinieren kann, um die Datenbankdurchläufe zu verringern.

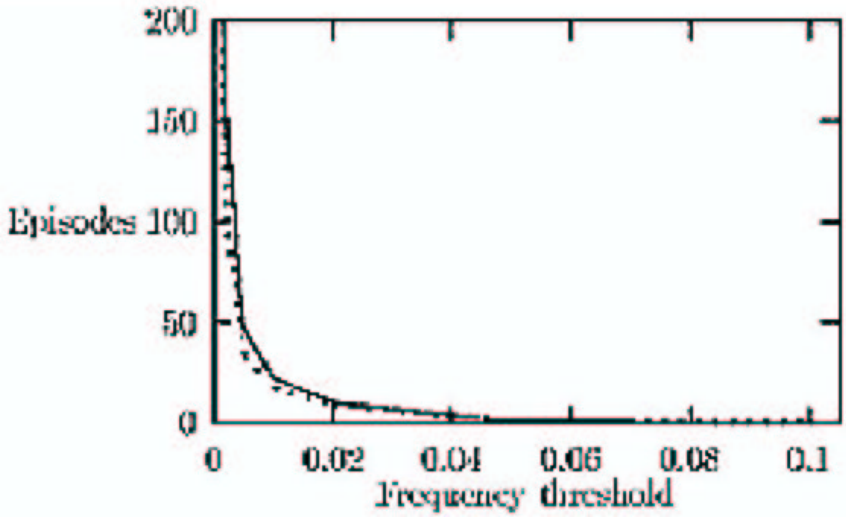


Abbildung 9.4: Anzahl serieller sowie injektiv paralleler Episoden

9.5.3 Vergleich der Algorithmen MINEPI und WINEPI

Die Tabellen 9.4 und 9.5 veranschaulichen die Performanzstatistiken beim Finden häufiger Episoden mit MINEPI. Diese Methode benutzt wie gesagt minimale Vorkommen. Vergleicht man diese Tabellen mit den Tabellen 9.1 und 9.2, so beobachtet man die gleiche Tendenz, dass die Anzahl der Kandidaten und Episoden rapide abnimmt, sofern der support threshold zunimmt.

Zwischen beiden Methoden gibt es einen Unterschied bei kleinen Episoden. Man stelle sich eine Episode a bestehend aus einem Event A vor. WINEPI entdeckt einen einzelnen Event A in 60 Fenstern, während MINEPI nur ein minimales Vorkommen sieht.

Die folgende Abbildung 9.6 zeigt den Zeitaufwand, um häufige Episoden mit MINEPI zu finden.

Der Zeitaufwand erreicht ein Plateau, ab dem sich die Grössen der maximalen Episoden nicht mehr ändern (hier ab einem support threshold von 500).

9.5.4 Regeln

Wie bereits erwähnt werden Regeln erstellt, indem alle häufigen Episoden a als rechte Seite der Regel und alle Subepisoden $b \leq a$ als linke Seite der Regel betrachtet werden. Die folgenden Tabellen 9.6 und 9.7 zeigen die Ergebnisse für serielle Episoden.

Episodengrösse	Episoden	Kandidaten	Häufige Episoden	Übereinstimmung
1	287	287	58	20%
2	82369	3364	137	4%
3	$2 * 10^7$	719	46	6%
4	$7 * 10^9$	37	24	64%
5	$2 * 10^{12}$	24	17	71%
6	$6 * 10^{14}$	18	12	67%
7	$2 * 10^{17}$	13	12	92%
8	$5 * 10^{19}$	13	8	62%
9	$1 * 10^{22}$	8	3	38%
10	$4 * 10^{24}$	3	2	67%

Tabelle 9.3: Verhalten bei unterschiedlichen Episodengrössen

Supportschwellenwert	Kandidaten	Häufige Episoden	Iterationen	Zeit (in s)
50	12732	2735	83	28
100	5893	826	71	16
250	2140	298	54	16
500	813	138	49	14
1000	589	92	48	14
2000	405	64	47	13
4000	352	53	46	12

Tabelle 9.4: Serielle Episoden

Supportschwellenwert	Kandidaten	Häufige Episoden	Iterationen	Zeit (in s)
50	10041	4856	89	30
100	4376	1755	71	20
250	1599	481	54	14
500	633	138	49	13
1000	480	89	48	12
2000	378	66	47	12
4000	316	53	46	12

Tabelle 9.5: Parallele Episoden

Supportschwellenwert	Eindeutige Regeln	Zeit zur Erzeugung der Regeln (in s)
50	50470	149
100	10809	29
250	4041	20
500	1697	16
1000	1221	15
2000	1082	14
4000	1005	14

Tabelle 9.6: Unterschiedlicher Supportschwellenwert

Anzahl Zeitschranken	Alle Regeln	Zeit zur Erzeugung der Regeln (in s)
1	1221	13
2	2488	13
4	5250	15
10	11808	18
20	28136	22
30	42228	27
60	79055	43

Tabelle 9.7: Unterschiedliche Zeitschranken

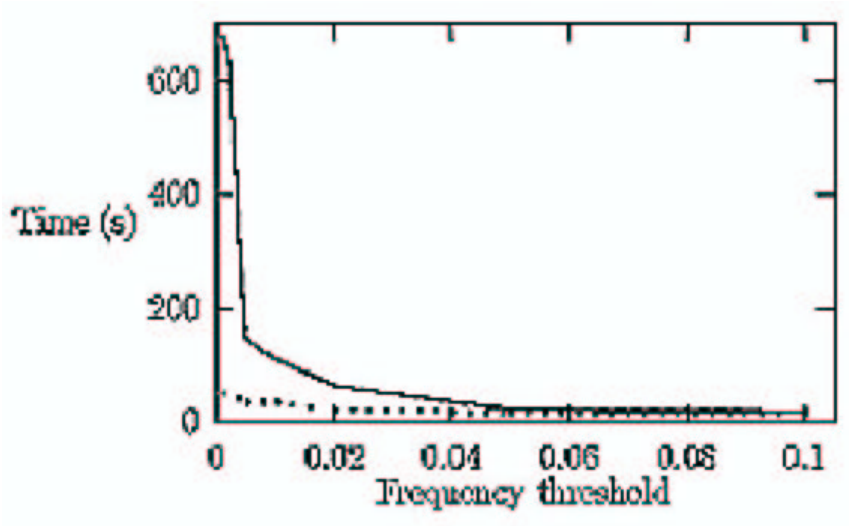


Abbildung 9.5: Bearbeitungszeit für serielle(durchgezogene Linie) und injektiv parallele(gepunktete Linie) Episoden mit WINEPI

Die Tabelle zeigt die Anzahl der Regeln, die von MINEPI mit einem confidence threshold Wert von 0 und einer maximalen Zeitgrenze von 60 s erkannt werden. Auf der linken Seite variiert der support threshold. Auf der rechten Seite werden Ergebnisse mit einer variablen Anzahl von Zeitgrenzen präsentiert.

Fast 80000 Regeln sind eine ganze Menge. Es gibt ja wohlgemerkt nur 1221 klare Regeln. Der Rest der Regeln ergibt sich aus verschiedenen Kombinationen der Zeitgrenzen. Verschiedene Kriterien können angewandt werden, um die interessantesten Regeln auszuwählen.

9.6 Abschluss

Es wurde eine Methode vorgestellt, um häufige Episoden in zeitlichen Daten zu erkennen. Die Methode bestand darin, Episoden als als partiell geordnete Reihen von Events zu betrachten, und auf Teile (Fenster) der Folge zu schauen.

Es wurde WINEPI beschrieben, um alle Episoden einer gegebenen Klasse von Episoden zu finden, die häufig (frequent) genug waren. Der Algorithmus fusst auf der Erkennung einer Episode als frequent, wenn alle ihre Subepisoden frequent sind. Weiterhin benutzt der Algorithmus inkrementelles Prüfen, ob eine Episode in einem Fenster erscheint.

Desweiteren wurde eine Alternative präsentiert: MINEPI, für die Erkennung häufiger Episoden, basierend auf minimalen Vorkommen von Episoden. Da

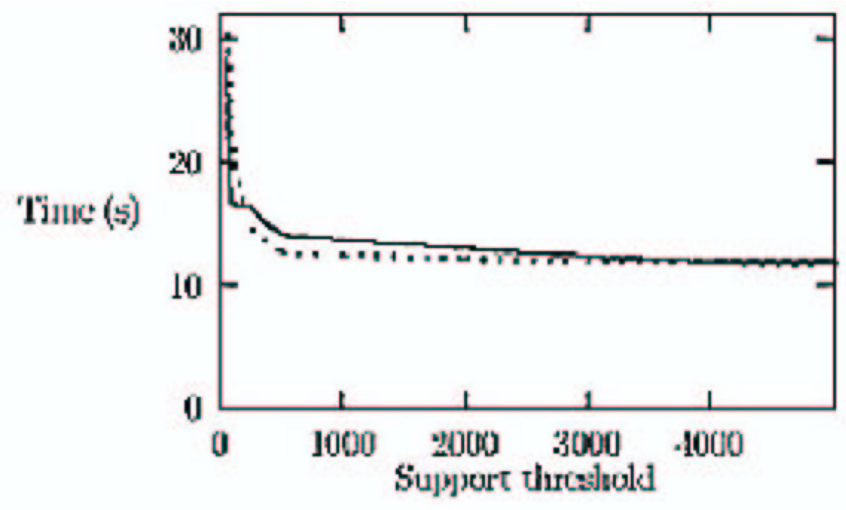


Abbildung 9.6: Bearbeitungszeit für serielle(durchgezogene Linie) und injektiv parallele(gepunktete Linie) Episoden mit MINEPI

diese Methode Regeln mit zwei Zeitschranken erstellen kann, ist sie mächtiger, um Verbindungen zwischen Events zu repräsentieren.

9.7 Bezug zu anderen Themen

In Kapitel 2: „Der KDD Prozess“ wurde das Verfahren der Wissensentdeckung in Datenbanken erklärt.

Vorgestellt wurde die Hierarchie sowie die Methodik des Prozessmodells in einem KDD-Cup. Vor allem das grafische Modell des KDD-Prozesses stellte eine gute Grundlage für das Verständnis der KDD-Welt dar. Vom Datenverständnis über die Datenvorbereitung bis hin zum DataMining sowie der Bewertung der Daten sollten alle angeführten Punkte in späteren Vorträgen wieder auftauchen. Aber vor allem die Phase des DataMining wurde ausführlich in allen Beiträgen behandelt. Da es sich bei diesem Beitrag um einen allgemeinen Bericht handelt, kann man keinen Bezug zu meinem Bericht nehmen.

Das Thema „Apriori für zeitliche Daten“ ist zum Beispiel ein Thema, das in den Bereich des DataMining einzuordnen ist. Folgendes Kapitel 3: „Handwerkszeug der Statistik“ behandelte statistische Methoden, die dazu dienlich sind, verschiedene DataMining-Verfahren zu vergleichen, indem die Daten oder Ergebnisse statistisch analysiert werden. Dieser Beitrag ist so gut wie für jeden Beitrag wichtig. Da für verschiedene Aufgaben verschiedenste Verfahren mehr oder weniger gut anzuwenden sind, benötigt man gewisse handwerkliche

Grundlagen, um die richtigen Verfahren zu erfassen.

Es folgten Berichte über verschiedene DM-Verfahren wie Entscheidungsbaeume (Kapitel 4) sowie Support Vector Machines (Kapitel 5) und Regellerner (Kapitel 6). Die grössten Gemeinsamkeiten zu meinem Thema haben aber wohl die Kapitel Subgruppenentdeckung (7) und natürlich Apriori (8). Ähnlich dem Entdecken von Episoden beim Apriori wird, wie der Name schon sagt, auch bei der Subgruppenentdeckung nach signifikanten Untergruppen gesucht. Und der Bezug zu Apriori ist natürlich unbestritten: bei WINEPI werden die signifikanten Daten noch mit Hilfe eines Fensters gefunden, das sich über den zeitlichen Ablauf der Daten bewegt. Beim einfachen Apriori versucht man auch häufige Ereignisse zu erkennen, im Endeffekt werden sogar gleiche Verfahrensweisen benutzt. Allein die Tatsache, dass es sich bei diesem Beitrag um zeitliche Daten handelt, ist als Unterschied der beiden Themen anzumerken.

Auch das Kapitel 10: Clustering kann mit Apriori sehr gut verglichen werden. Es werden wiederum Teilgruppen gesucht, die aussagekräftige Informationen liefern. Beim „Bagging und Boosting“ (Kapitel 11) verhält es sich etwas anders, da hier der zu behandelnde Datensatz zuerst einmal „manipuliert“ wird, in Bezug kann man dieses Verfahren zu Apriori für zeitliche Daten also nicht setzen. Die Merkmalsgenerierung (Kapitel 12) führte noch einmal vor Augen, dass es sinnvoll sein kann, gegebene Daten vorzubehandeln. Allerdings ist dies eher eine Methode, die man vor dem DataMining durchführt, auch hier kann man einen Bezug nicht direkt herstellen.

Das abschliessende Kapitel 13: „Die Welt des KDD Cup“ war ein guter Schlussvortrag, da viele kennengelernte Methoden in ihrer Anwendung erwähnt wurden. Auch konnte man einschätzen, inwiefern eine KDD-Aufgabe zu bewältigen ist. Dieser Beitrag kann also sozusagen zu fast allen anderen in Bezug gesetzt werden.

Kapitel 10

Clustering

Anna Litvina

10.1 Einführung zum Clustering

10.1.1 Problem Definition

Wenn man mit einer sehr großen Anzahl der Instanzen umgehen muss, bevorzugt man häufig mit einer kleineren Menge von homogenen Gruppen der Instanzen sich zu befassen. Wenn so eine Menge von Gruppen entdeckt ist, so untersucht man die Gruppen genauer auf gemeinsame Eigenschaften (z.B. gemeinsame Ziele oder gemeinsame Ursachen). Das dient sowohl zur Vereinfachung von Kommunizieren als auch zum kleineren Verbrauch von Speicherplatz. Das findet heutzutage die Anwendung in sehr vielen Bereichen: Mustererkennung, Marktforschung, Bildverarbeitung, Datenanalyse etc.
Man definiert Clustering als Prozess vom Organisieren von Objekten von einem Instanzenraum in Gruppen, so dass die Objekte einer Gruppe (eines Clusters) gewissermaßen ähnlich zu einander sind und dass die Objekte verschiedener Gruppen möglichst unähnlich sind. Meistens wird es verlangt, dass alle Cluster eine Partitionierung vom Instanzenraum darstellen, d.h. der Durchschnitt je zwei beliebiger Cluster eine leere Menge ist und die Vereinigung aller Cluster das Instanzenraum selber ist.
Beim konzeptuellen Clustering wird es versucht nicht nur die Cluster zu bilden, sondern auch eine charakteristische und kurze Beschreibung für jedes Cluster zu beschaffen.
Es gibt eine lange Reihe von verschiedenen Cluster-Verfahren, klassischen und neuartigen, die man oft kombiniert und modernisiert. Die zwei älteren und am häufigsten Verfahren sind partitionierendes (k-Clustering) und hierarchisches Clustering. Bei dem k-Clustering wird jedes Objekt genau zu einer Gruppe zu-

geordnet. Während bei dem hierarchischen Clustering wird jede Gruppe von Objekten aus kleineren Gruppen zusammengestellt. Man unterscheidet auch Dichte-basierte Methoden, Gitter-basierte Methoden, Modell-basierte Methoden und viel mehr.
Hier wird versucht die fundamentalen Ideen von Clustering-Algorithmen vorzustellen, die Problematik von den Hauptansätzen zu beschreiben und einige von den bekanntesten Methoden zusammenzufassen.

10.1.2 k-Clustering

Im Allgemeinen nehmen k-Clustering Algorithmen als Eingabe eine Menge S von Objekten und eine ganze Zahl k und geben als Ausgabe eine Partition von S in Untermengen $S_1, S_2, \ldots, S_k$. Der häufigste Typ von k-Clustering Algorithmen ist ein Optimisierungsalgorithmus. Typischerweise wird angenommen, dass die Elemente von S aus dem d-dimensionalen Raum sind, und dass es eine Kostenfunktion c gibt, die jedem Cluster gewisse Kosten zuordnet.
Die Hauptidee ist die Summe der Kosten über allen Cluster zu minimieren:

$$\min \sum_{i=1}^{k} c(S_i)$$

Das häufigste Kriterium dabei ist das *"sum-of-squares"* Kriterium, das die Kostenfunktion als

$$c(S_i) = \sum_{r=1}^{|S_i|} \sum_{s=1}^{|S_i|} (d(x_r^i, x_s^i))^2$$

definiert, wobei x_r^i das $r$te Element von S_i, $|S_i|$ die Anzahl der Elemente in S_i und $d(x_r^i, x_s^i)$ die Distanz zwischen x_r^i und x_s^i sind. In mehreren Runden werden die Elemente jeweils dem nächsten Cluster zugeordnet, das durch sein repräsentatives Element, seinen Mittelwert oder Median bzw. Medioid repräsentiert wird. Im ersten Fall, spricht man über k-means Algorithmus mit der Kostenfunktion

$$c(S_i) = \sum_{r=1}^{|S_i|} (d(\hat{x}^i, x_s^i)),$$

wo $\hat{x}^i$ der Mittelwert (centroid) von dem $i$te Cluster ist. Voraussetzung dabei - eine Möglichkeit den Mittelpunkt zu finden. Im zweiten Fall, spricht man über k-medioids Algorithmus, mit der Kostenfunktion

$$(S_i) = \sum_{r=1}^{|S_i|} (d(\bar{x}^i, x_s^i)),$$

wo $\bar{x}^i$ der Medianwert von dem $i$te Cluster ist. Die Skizze für die Methode könnte beispielsweise so aussehen:

(1) Wähle zufällig k Objekte als initiale Mittelwerte (oder Mediode) der Cluster (2) Wiederhole (3) und (4), solange es eine Verbesserung gibt: (3) Ordne jedes Objekt einem Cluster zu (gemäß Ähnlichkeitsfunktion oder nächstem Mediod) (4) Aktualisiere die Cluster-Mittelwerte (oder Mediode)
Skizze 1: k-Partitioning Algorithmus

Alternativ kann man $\max_{1 \leq i \leq k} c(S_i)$ minimieren, mit der folgenden Kostenfunktion:

$$c(S_i) = \max_{x_r^i, x_s^i \in S_i} d(x_r^i, x_s^i)$$

Die Formulierung ist von der Bedeutung, weil es eine leichte 2-Approximation für die Lösung gibt.
Dieser Ansatz hat leider manche Schwächen. Erstens, begünstigen die Algorithmen nur sphärische (kugelförmige) Cluster und erlauben die komplexeren Cluster Formen nicht. Zweitens, reagieren die Algorithmen sensibel auf "Rauschen". Außerdem, muss die Anzahl der Cluster vorgegeben werden.

10.1.3 Hierarchisches Clustering

Das Ziel des hierarchischen Clustering ist eine baumartige Struktur $T(S)$ zu konstruieren, wo jeder Knoten eine Untermenge von S repräsentiert. Also, die Wurzel des Baumes enthält alle Elemente von S und jedes Blatt des Baumes fasst ein Element der Menge S um. Geht man durch den Baum runter, so besucht man die Mengen von Elementen, die mit der Tiefe des Baumes zunehmend ähnlicher werden. Eine Partition von S in Cluster ist eine beliebige Menge von Knoten, so dass jeder Pfad von einem Blatt zur Wurzel auf genau ein Element aus der Menge trifft.
Man unterscheidet zwei Methoden vom hierarchischen Clustering: aufteilende Algorithmen und aggregierende Algorithmen. Die ersten teilen die Menge S rekursiv auf, bis die einelementigen Untermengen erreicht sind. Die aggregierende Algorithmen platzieren die einzelnen Elementen in den einelementigen Untermengen und vereinigen jeweils die zwei Untermengen, die nach der Kostenfunktion am billigsten zu vereinigen sind. Der Prozess wird rekursiv wiederholt bis eine einzige Menge geblieben ist. Die Varianten unterscheiden sich lediglich in der Kostenfunktion, die entscheidet, welche Untermengen zum Verbinden gewählt werden, die üblichen Maßen dafür sind [19]: single-link mit

$$c(S_i, S_j) = \min_{x_i \in S_i, x_j \in S_j} d(x_i, x_j),$$

average-link mit Kostenfunktion

$$c(S_i, S_j) = \frac{1}{|S_i||S_j|} \sum_{x_i \in S_i} \sum_{x_j \in S_j} d(x_i, x_j),$$

und complete-link mit

$$c(S_i, S_j) = \max_{x_i \in S_i, x_j \in S_j} d(x_i, x_j),$$

Die größten Schwächen dieser hierarchischen Clustering Methoden sind ähnlich denen, vom k-Clustering: average-link und total-link begünstigen nur sphärische Cluster, während single-link Clustering unerwünscht verlängerte Cluster produzieren kann. Falls ein Zerlegungs- oder Verschmelzensschritt vorgenommen worden, so kann man in dem Verfahren nichts rückgängig machen.
Man kann die Cluster bei hierarchischem Clustering auch mit Hilfe der Wahrscheinlichkeit beschreiben. Der Ansatz wurde im COBWEB [28] benutzt und wird im Kapitel 9.2.5 ausführlicher vorgestellt.

10.1.4 Eingabe

Im Grunde, operiert jeder Clustering Algorithmus über eine Menge von Eingabevektoren $x_1, x_2, \ldots, x_n$ im d-dimensionalen Raum. Die Daten können als Datenmatrix (Objekt-Attribut Struktur) präsentiert werden. In der Praxis benutzen aber die Algorithmen die Eingabevektoren nicht direkt. Als Eingabe benutzt man oft Ähnlichkeitsmatrizen (Objekt-Objekt Struktur), wo man den Grad der Ähnlichkeit zwischen allen Elementen speichert. Alternativ benutzt man auch Distanzmatrizen, die den Abstand zwischen Elementen repräsentieren. Wahl des richtigen Ähnlichkeitsgrades bzw. der Metrik für den Abstand ist aber aufwendig. Beispielsweise gibt es mehrere Möglichkeiten für die Berechnung der Ähnlichkeitsmaße zwischen zwei beliebigen Objekten im p-dimensionalen Raum: Euklidische Distanz und Manhattan Distanz, wobei die Attribute jeweils noch unterschiedlich gewichtet werden können.
Einige Methoden, z.B. single-link hierarchische Clustering verlangen als Eingabe nur den Vektor und eine Funktion, die den nächsten Nachbar liefert.
Auch die Eingaberäume (numerische, binäre, nominale, ordinale usw.) können verschiedene mathematische Eigenschaften haben. Das beeinflusst stark den Clustering-Vorgang und die Wahl der Methode.

10.1.5 Von Cluster zum Begriffsbeschreibung

Die bloße Gruppierung der Objekte ist für sich selbst genommen nicht so informativ und nützlich. Nehmen wir an, wir haben ein Clustering-Verfahren durchgeführt, alle vorhandenen Objekte klassifiziert und danach kommt ein

neues Objekt, das wir auch klassifizieren müssen. Logischerweise muss man nicht den ganzen Clustering-Vorgang wiederholen, sondern muss man die vorhandenen Ergebnisse benutzen können. In dem Fall benutzt man den Begriff vom konzeptuellen Clustering.
Beim konzeptuellen (oder begrifflichen) Clustering wird auch nach einer Beschreibung der Gruppe gesucht. Je einfacher und allgemeiner diese Beschreibung ist, desto besser ist üblicherweise die Qualität der Cluster und deren Repräsentation. Folglich kann man neue Objekte schneller und einfacher klassifizieren.
Es gibt mehrere Alternativen ein Cluster zu repräsentieren. Erstens, durch einen Repräsentanten: das erste Objekt, ein "typisches" Objekt oder einen "Durchschnittsfall". Weiterhin, kann man ein Cluster als Konzept repräsentieren, z.B. durch eine Menge von Bedingungen, die notwendig oder hinreichend sind, oder das Konzept kann aus einer Wahrscheinlichkeitsverteilung aller Attributwerte des Clusters bestehen.

10.2 Neuartige Modelle und Vorgänge

10.2.1 Mixture Modelle

Dieses Verfahren wurde von Banfield and Raftery präsentiert [19]. Die grundlegende Idee der Mixture Modelle ist folgende: die Eingabevektoren $x_1, x_2, \ldots, x_n$, sind Beobachtungen von einer Menge von k unbekannten Verteilungen $E_1, E_2, \ldots, E_k$. Die Dichte einer Beobachtung x_r ist durch $f_i(x_r|\theta)$ gegeben, wobei θ eine unbekannte Menge der Parameter ist. [61] Nehmen wir an, τ_r^i repräsentiert die Wahrscheinlichkeit, dass x_r zu E_i gehört, dann ist $\sum_{i=1}^{k} \tau_r^i = 1$, weil jede Eingabe zu einer Verteilung gehören soll. Das Ziel der Schemata ist, solche Parameter θ und τ zu finden, die die Wahrscheinlichkeit maximieren:

$$L(\theta, \tau) = \prod_{r=1}^{n} \sum_{i=1}^{k} \tau_r^i f_i(x_r|\theta)$$

Wenn man das Modell so einschränkt, dass jede Eingabe zu genau einer Verteilung gehört, dann sieht die zu maximierende Wahrscheinlichkeit anders aus:

$$L(\theta, \gamma) = \prod_{r=1}^{n} f_{\gamma_r}(x_r|\theta),$$

wobei $\gamma_r = i$ wenn x_r zu E_k gehört. Das Modell erlaubt auch Rauschen explizit zu repräsentieren, z.B. als Poisson Verteilung mit $\gamma_r = 0$, wenn x_r ein "verrauschter" Punkt ist und E_0 die Menge aller verrauschter Punkte sind.

Beim Modellieren der Verteilungen als mehrdimensionale normalverteilte Funktion sind die unbekannten Parameter τ die Mittelwertvektoren μ_i und Kovarianzmatrizen Σ_i für jede Verteilung. Die Werte von μ_i,Σ_i und γ sind dann mit Hilfe des üblichen iterativen Expectation-Maximization (EM) Prozesses berechenbar. Wenn aber die Parameter für jede Verteilung beliebig variieren dürfen, dann kann die Suche des globalen Optimums für die großen Probleme zu teuer werden. Eine mögliche Lösung des Problems wäre Σ_i zu zerlegen als $\Sigma_i = D_i \Lambda_i D_i^T$, wobei D_i die Matrix der normalisierten Eigenvektoren und Λ_i die Diagonalmatrix der Eigenwerte von Σ_i sind. Unter dieser Parametrisierung, die Orientierung von E_i ist durch D_i bestimmt und die Form und Dichte Konturen von E_i sind durch Λ_i bestimmt. Der Ansatz erlaubt es manche sehr nützliche Statistiken abzuleiten, die für weitere Analysen hilfreich sein können. Z.B. approximate weight of evidence (AWE), das die Bayessche a posteriori Wahrscheinlichkeit bestimmt, die man zum Vergleich der Mixture Modelle und k-Werte benutzen kann [19].
Der Ansatz bleibt aber immer noch relativ unbeliebt im Gegensatz zu den klassischen Arten des Clustering. Mögliche Gründe dafür sind, dass der Algorithmus nicht auf Effizienz (Laufzeit) fokussiert. Es gibt derzeit keine Effizienzanalyse des Algorithmus. Zweitens, sind aufwendige manuelle Eingriffe in den Algorithmus erforderlich. Das betrifft sowohl die Bestimmung des Modells als auch das Bestimmen der Anzahl der Cluster. Drittens, basieren Mixture Modelle sich auf der Annahme, dass die Daten der Gauss-Verteilung entsprechen. Das ist aber nicht immer der Fall. Noch schlimmer ist dass der Algorithmus nur mit numerischen Daten funktioniert, kategorische Daten kann man damit nicht verarbeiten.

10.2.2 Dynamische Systeme

Gibson, Kleinberg and Raghavan präsentierten den folgenden Ansatz [20]. In der Praxis, stößt man sehr häufig auf kategorische Daten, bei denen jedes Element Attribute hat, die Werte aus festen kleineren Mengen annehmen können. Es gibt weder Distanzen noch ein Maß der Ähnlichkeit zwischen den Werten. Man sieht nur, ob die Elemente den gleichen Wert in einem bestimmten Feld haben oder nicht.
Die Idee des Verfahrens ist das Clustering-Problem auf einen Graphen zu transformieren. Stark zusammenhängende Komponenten im Graphen präsentieren dann ein mögliches Clustering. Nehmen wir an, wir haben als Eingabe eine Menge $T = \{t_1, t_2, \ldots, t_n\}$ von Tupeln mit d Feldern. Man betrachtet die Eingabe als einen Graphen, bei dem die Knoten $V = \{v_1, v_2, \ldots, v_m\}$ die Menge aller Werten darstellen, die in den Eingabetupeln auftauchen. Jedes Tupel ist dann ein Pfad im Graphen. Außerdem wird mit jedem Knoten v ein Gewicht ω_v assoziiert. Der Vektor w, der alle Gewichte von allen Knoten enthält, heißt Konfiguration des Graphen.

Im Kern des Ansatzes steht eine Funktion f, die die gegenwärtige Konfiguration auf die nächste abbildet (Gewichte aller Knoten werden aktualisiert). Die Idee ist, dass die Funktion so gewählt ist, dass die Knoten, die ein größeres Gewicht und mehr Nachbarn-Knoten haben, größer gewichtet werden. Der Prozess wird mehrmals wiederholt bis ein Fixpunkt erreicht wird. Man kann die Funktion wie folgt definieren:

(1) Für jeden Knoten v aktualisiere sein Gewicht ω_v wie folgt:
für jedes Tupel $t = \{v, u_1, \ldots, u_{d-1}\}$
$x_t \longleftarrow \oplus(\omega_{v_1}, \ldots, \omega_{v_{d-1}})$
$w_v \longleftarrow \sum_t x_t$

(2) Normalisiere die Konfiguration so, dass die Summe von Quadraten der Gewichte in jedem Feld 1 ist

Skizze 2: Funktion zur Abbildung auf die nächste Konfiguration

Als Kombinationsoperator $\oplus$ kann man eine Reihe der Operatoren wählen, z.B. Produkt, Summe, maximaler Wert, Mittelwert usw.
In der Praxis, für eine beliebige Konfiguration konvergiert die Funktion f gegen einen Fixpunkt. Für zwei Mengen von Tupeln, die keine Felder mit denselben Werten haben, wächst die Wahrscheinlichkeit, dass die Mengen "getrennt" werden, desto schneller je mehr die Mengen ungleich in der Größe sind. Für die Systeme mit mehreren Fixpunkten, man kann relativ schnell alle Fixpunkte finden, indem man von verschiedenen Anfangskonfigurationen startet.
Der Ansatz hat auch einige Schwäche. Vor allem der Mangel an theoretischem und praktischem Verständnis des Algorithmus. Theoretisch ist die Konvergenz nicht mit jedem Kombinationsoperator garantiert und die Wahl der Operatoren bleibt jedes Mal das Problem. Der Ansatz ist zudem nur auf kategorischen Daten beschränkt.

10.2.3 Clique Graphs

Clique Graphs Verfahren wurde von Ben-Dor and Yakhini präsentiert [19]. Grundlegende Idee des Ansatzes ist das beschädigte Clique Modell. Idealerweise sind alle Elemente in einem Cluster sich einander ähnlich und den übrigen Clustern nicht ähnlich. In der Praxis ist die Ähnlichkeitsrelation nur approximativ. Bei einer Präsentation der Relation durch einen Graphen, sieht der Graph nicht als die Menge der disjunkte Cliquen, sondern "beschädigt". Im beschädigten Cliquen Modell, nimmt man an, dass die Eingabe eines Clique Graphen beschädigt durch Hinzufügen von einigen nicht existierenden Kanten und Löschen von einigen vorhandenen Kanten wurde, jeweils mit einer Wahr-

scheinlichkeit von α. Das Ziel ist dann, den Originalgraphen zu finden, gegeben das beschädigte Modell.
Der Hauptunterschied zum Graphen Aufbau bei den dynamischen Systemen (siehe 9.2.1) ist das hier jeder Knoten im Graphen ein Objekt darstellt, nicht aber den Attributenwert.
Es gibt mehrere Algorithmen um das originale Model zu rekonstruieren [19]. Ben-Dor and Yakhini behaupten, dass wenn man zufällig eine kleine Menge der Knoten wählt, zeigen die Knoten doch die Struktur, des ursprünglichen Clusterings. Formal kann man das Verfahren wie folgt beschreiben: nehmen wir an, V ist die Menge der Knoten in dem Eingabevektor, und V ist die Vereinigung von den disjunkten Mengen, so dass $V = E_1 \cup E_2 \cup \ldots \cup E_k$, wobei jedes E_i ein Cluster repräsentiert. Zudem, gibt es für jedes v ein $c(v) = i$, genau dann, wenn $v \in E_i$. $c(v)$ ist ein Label und beschreibt das Cluster, zu dem v gehört. Der Kern V' von V ist als $V' = E'_1 \cup E'_2 \cup \ldots \cup E'_k$ definiert, wobei E'_i eine nichtleere Untermenge von E_i ist. Der Kern V' klassifiziert die Eingabe $v \in V - V'$ mit Hilfe der folgenden Funktion:

$$c_{V'}(v) = \max_{1 \leq i \leq k} deg(v, E'_i)/|E'_i|,$$

wo $deg(v, E'_i)$ die Anzahl der Kanten aus E'_i ist, die aus dem Knoten v ausgehen. Intuitiv, repräsentiert $c_{V'}(v)$ das Cluster, das v zu enthalten scheint, nur auf der Ähnlichkeit von v zu den Elementen aus dem Kern basierend. Wegen $c_{V'}(v) = c_V(v)$ klassifiziert V' v korrekt.
Unter der Annahme, dass die Größe des kleinsten Clusters nicht zu gering ist und auch dass α nicht zu klein ist, enthält mit großer Wahrscheinlichkeit eine zufällige Untermenge von V einen Kern, der alle Eingaben korrekt klassifiziert. Der Algorithmus sieht wie folgt aus:

(1) Wähle zufällig eine Untermenge V' von V
(2) Betrachte alle möglichen Partitionierungen von V' in nichtleere Cluster. Jede Partitionierung ist ein Kandidat für den Kern
(3) Klassifiziere mit Hilfe von jedem Kandidat alle Punkten, die übrig geblieben sind. Behalte die Klassifikation, die in dem Clique Graphen resultiert, der dem Eingabe Graphen am ähnlichsten ist

Skizze 3: Clique Graphs Algorithmus

Falls die Cluster Anzahl unbekannt ist, kann man den Algorithmus etwas modifizieren. Als Eingabe braucht man eine boolesche Ähnlichkeitsfunktion s und einen Koeffizient t. Man bekommt die Affinität $a(v, E)$ vom Element v zu den Cluster E:

$$a(v, E) = \sum_{u \in E} s(u, v)$$

Ein Element hat hohe Affinität zu E, falls $a(v, E) \geq t|E|$. Der Algorithmus konstruiert ein Cluster nach dem anderen durch hinzufügen der Elemente mit hoher Affinität zu einen Cluster und Entfernen der Elemente mit niedriger Affinität.
Man berichtet [19], dass der Algorithmus sehr präzise ist. Leider fehlen derzeit Untersuchungen über Zeit- und Speicherplatz Anforderungen.

10.2.4 Subspace Clustering

Clustering in Unterräume ist ein weiterer innovativer Ansatz. Das folgende Verfahren wurde von Agrawal, Gehkre et al. präsentiert [60]. Nach dem Algorithmus wird Clustering unabhängig in jedem Unterraum durchgeführt. Danach wird eine kompakte Zusammenfassung für jedes Cluster in DNF zusammengestellt.
Nehmen wir an, die Eingabe ist eine Menge von n d-dimensionalen Vektoren aus dem Raum $A = A_1 \times A_2 \times \ldots \times A_d$, wobei A_i eine Wertemenge ist. Zudem wird jedes A_i in ξ Intervalle diskretisiert. Bei solch einer Diskretisierung wird der originale d-Raum in eine Menge von d-dimensionalen achsenparallelen Rechtecken, die man Regionen nennt. Eine Region heißt dicht, falls der Anteil der Eingabevektoren, enthalten in der Region, einen vom Benutzer angegebenen Parameter τ überschreitet. Wir betrachten einen Graphen, bei dem alle Knoten dichte Regionen sind und wo es eine Kante zwischen den Knoten gibt, falls die entsprechenden Regionen im Raum benachbart sind. Man kann dann ein Cluster als eine verbundene Komponente im Graphen definieren. Eine Region kann man als eine Konjunktion von Wertebereichen jeder Dimension auffassen, während man ein Cluster als eine Disjunktion von Konjunktionen von Regionen definiert, die sie umfassen.
Der Algorithmus hat aber leider sehr viele Nachteile. Die Parameter τ und ξ sind nicht leicht zu wählen. Z.B. verschiedene Cluster haben unterschiedliche "Dichte", d.h. der Parameter τ, der einer Region passt, kann einer anderen Region nicht passen. Analog, der Unterschied in der Verteilung in jeder Dimension verursacht unterschiedliche Werte für ξ.

10.2.5 Iteratives Optimieren im hierarchischen Clustering

Die Cluster können beim hierarchischen Clustering auch mit Hilfe der Wahrscheinlichkeit beschrieben werden [28], d.h. jeder Variablenwert hat eine zugehörige bedingte Wahrscheinlichkeit, $P(A_i = V_{ij}|C_k)$, die das Verhältnis von den Beobachtungen in C_k widerspiegelt, die den Wert V_{ij} entlang der Variable A_i haben, wie z.B. im COBWEB [28]. Tatsächlich ist die Wahrscheinlichkeit durch

die Anzahl von Beobachtungen im Cluster, bei denen Variable A_i Wert V_{ij} hat, gegeben. In der Art beschriebene Cluster, die in eine hierarchische baumartige Struktur organisiert sind, sind als *"probablistic-categorization"* Baum bekannt. Diese Art der Beschreibung der Daten erlaubt ein robustes Clustering-Verfahren, das hierarchisches Sortieren genannt ist. Das Verfahren bildet rekursiv eine Baumstruktur. Gegeben eine aktuelle hierarchische Struktur und ein neues zuklassifizierendes Element. Man berechnet die Qualität des neuen Clusterings, das aus dem Hinzufügen des Elements zu jedem Cluster und zu einem neuen Cluster resultiert. Die Option, die als die beste bewertet wird, wird ausgewählt. Zudem kann die Tiefe des Baums fixiert sein. Das resultierende Clustering hängt von der Reihenfolge in der die Objekte zugefügt werden ab. Das ist eine der Schwächen hierarchischen Sortierens. Trotzdem kann eine Anfangsstruktur von Objekten mit hierarchischem Sortieren generiert werden, die später verbessert werden kann. Eine Möglichkeit bieten iterative Optimierungsverfahren. Die Strategie der iterativen Optimierung erlaubt eine Suche durch den Raum von Clusterings, die mit einer Anfangseinteilung in Cluster anfängt und wiederholt neue Cluster konstruiert, die laut der Zielfunktion besser sind, und die man relativ billig konstruieren kann.
Es gibt eine Reihe von Heuristiken zum iterativen Optimieren. Wir unterscheiden hier die drei wichtigsten [28]. Erstens, Reorder und Resorting; zweitens, iteratives Redistribution einzelner Objekte und drittens, iteratives hierarchisches Redistribution.
Reorder und Resorting: Die Idee, die im Verfahren benutzt wird, basiert auf dem Fakt, dass hierarchisches Sortieren wesentlich besser funktioniert, wenn die Objekte, die am Anfang zugefügt werden, aus verschiedenen Bereichen sind, also möglichst unähnlich sind. Nach dem Verfahren werden die Objekte gemäß einer beliebigen Ordnung sortiert, danach wird eine "Unähnlichkeits" - Reihenfolge aus dem Clustering konstruiert, und die Objekte werden entsprechend der neuen Reihenfolge sortiert. Es wird dabei rekursiv eine Liste von Objekten von den am wahrscheinlichsten zu den am wenigstwahrscheinlichen konstruiert. Danach werden die Listen gemischt und vor dem Ende der rekursiven Aufrufe werden die Objekten vom wahrscheinlichsten Cluster am Anfang platziert. D.h. nach der Anfangseinteilung in Cluster wird die "Unähnlichkeits"-Reihenfolge ausgefiltert, die Objekte werden umorganisiert. Das terminiert bis keine weitere Verbesserung in Qualität gefunden wird.
Die Idee von dem iterativen Redistribution ist ganz einfach: die Objekten im einstufigen Clustering werden aus ihren originalen Clustern entfernt und neu sortiert. Der Prozess terminiert bis zwei aufeinander folgende Iterationen dasselbe Clustering ausgeben. Die Strategie ist ganz robust, aber hat auch begrenzte Möglichkeiten lokalen Maxima zu entkommen.
Iteratives hierarchisches Redistribution ist ein Schritt weiter verglichen mit dem Redistribution einzelner Objekte: dabei werden nicht die einzelnen Objekten neu verteilt, sondern eine Menge von Objekten. Gegeben sei ein hierarchi-

sches Clustering, in einer rekursiven Schleife untersucht man alle Geschwister-Cluster durch eine Tiefensuche. Für jede Menge von Geschwistern untersucht man eine innere iterative Schleife die Geschwister, entfernt sie von ihrer aktuellen Position in der Hierarchie und findet für den Knoten mit seinem Unterbaum einen neuen Platz.
Der Prozess kann positiv erweitert werden, indem man einige zusätzliche Operatoren benutzt, z.B. Mischen, Zersplittern und Promotion. Beim Mischen werden zwei Geschwister-Cluster zusammengemischt, falls das die Qualität der Partition, zu der die Cluster gehören, verbessert. Zersplittern entfernt ein Cluster und setzt seine Kinder nach oben in der Hierarchie. Der Promotion Operator setzt ein Objekt auf die nächst höhere Stufe in der Hierarchie.
Eine Reihe von Experimenten wurde mit allen oben genannten Methoden des hierarchischen Clustering durchgeführt [28]. Dabei benutzte man verschiedene Datenbanken und verwendete sowohl zufällige als auch nicht zufällige Reihenfolgen von Objekten beim Generieren von Ausgangseinteilungen in Cluster von Objekten. Die wichtigsten Entdeckungen bei den Experimenten waren: erstens, das hierarchische Sortieren von zufälligen Eingaben gibt immer ein gutes Ergebnis. Zweitens, das hierarchische Redistribution funktioniert im Allgemeinen besser als alle anderen Methoden. Drittens, Reorder und Resorting kommt in den meisten Fällen näher an die Resultaten des hierarchischen Sortierens. Viertens, Redistribution einzelner Objekten kann den Anfangszustand nur wenig verbessern und ist wesentlich schlechter als die beiden anderen Verfahren. Laufzeiten waren am besten beim anfänglichen hierarchischen Sortieren. Das hierarchische Redistribution war wieder besser als die anderen zwei Methoden. Trotzdem wird die Laufzeit beim hierarchischen Redistribution schlechter mit größerer Tiefe des Baumes.
Wichtig ist, dass keine der drei Techniken abhängig vom Anfangssortieren oder von der Zielfunktion ist. Das erlaubt z.B. das Benutzen einer beliebigen agglomerativen Strategie anstatt dem hierarchischen Sortieren als Vorverarbeitung der Daten. Trotz dem Fakt, dass die Verfahren unabhängig von der Zielfunktion benutzt werden können, wird das hierarchische Redistribution durch das Benutzen von average PU (Partition Utility), das im COBWEB benutzt wurde [28],

$$PU = \sum_k P(C_k \sum_i \sum_j [P(A_i = V_{ij}|C_k)^2 - P(A_i = V_{ij})^2]/n$$

sehr begünstigt. Das spricht natürlich für das hierarchische Redistribution, macht aber den Vergleich der drei Methoden unmöglich.
Hierarchisches Clustering produziert eine Hierarchie mit der großen Anzahl der Clustering. Man will oft aber eine flache Struktur haben, also muss man entscheiden wie tief im produzierten Baum sein gewünschtes Clustering liegt. Die "Grenzen" für das Abschneiden des Baumes können durch das Resampling festgestellt werden. Nämlich wird die passende "Grenze" für jede Variable ge-

sucht. Solche "Grenzen" können für zwei beliebige Variablen verschieden sein und folglich die Cluster innerhalb dieser "Grenzen" beliebig tief liegen können. Das Verfahren der Suche nach den "Grenzen" ist ganz einfach: für jede Variable werden die Objekten aus der Validierungsmenge durch hierarchischem Clustering klassifiziert, dabei wird der Wert der Variable mit "maskiert" benutzt. Für jedes begegnete Cluster wird der Wert der Variable mit der am wahrscheinlichsten Wert der Variable für dieses Cluster verglichen, falls die gleich sind, dann wurde der Wert von der Variable für das Cluster korrekt klassifiziert. Die Anzahl von den richtig klassifizierten Werten für jedes Cluster wird notiert. Nach dem das Verfahren für alle Variable von allen Objekten gemacht wird, wird die beste "Grenze" für jede Variable, die die Anzahl der korrekten Klassifizierungen maximiert, gewählt. Die Strategie erleichtert wesentlich das Verfahren von hierarchischem Sortieren, z.B. ein Knoten, die unter der "Grenze" von allen Variablen liegt, reflektiert keine anständige Struktur und kann abgeschnitten werden.
Das oben genannte Verfahren wurde ganz ausführlich von den Wissenschaftlern getestet [28]. Das wichtigste Ergebnis dieser Tests ist, dass die Validierung und Pruning wesentlich verkleinern die Größe des Clusterings; und das vermindert die Genauigkeit nicht.

10.3 Analyse

10.3.1 Wahl der Methode

Alle oben beschriebene Verfahren sind für sich genommen nicht besonders schwierig. Merkwürdigerweise ist die schwierigste Aufgabe beim Clustering die Wahl der Methode für ein bestimmtes Problem [19]. Unten wird versucht eine Liste von wichtigsten Kriterien zusammenzufassen. Trotzdem, wird die Wahl von der passenden Methode eine Anzahl von tradeoffs beinhalten.

- Daten Modell. Ein sehr wichtiges Kriterium bei der Methoden Auswahl. Wie oben gesagt, viele Algorithmen erlauben nur bestimmte Eingaben, numerische (wie z.B. k-means Methode) oder kategorische (Dynamische Systeme). Gemischte Daten werden nur von wenigen Algorithmen akzeptiert (Subspace Clustering und Clique Graphs). Einige Methoden verlangen außerdem eine Distanz Metrik oder ein Ähnlichkeitsmaß.

- Cluster Gestalt. Sehr oft gibt es wenig a priori Wissen über die Gestalt der Cluster. Doch manche Methoden funktionieren nur bei der gewissen Form der Cluster gut. Beispielsweise, behandeln Mixture Modelle sphärische und ellipsoide Cluster am besten. Bei manchen Verfahren sollen

die Cluster gut getrennt sein (single-link hierarchisches Clustering). Also ohne Vorwissen ist es schwierig die richtige Entscheidung zu treffen.

- Skalierbarkeit. Beim Data-Mining befasst man sich üblicherweise mit großen Datenmengen, deshalb müssen die Methoden strengeren Anforderungen an die Rechenzeit und den Speicherplatzbedarf entsprechen. Leider sind noch nicht alle beschriebene Methode auf ihre Skalierbarkeit untersucht worden (Mixture Modelle und Clique Graphs).

- Rauschen. Nicht alle Methoden können das Problem von Rauschen behandeln. Beim Clique Graphs Modell werden verrauschte Punkte zu einzelnen winzigen Cluster. Dynamische Systeme ignorieren solche Punkte einfach, weil sie aus "dichten" Regionen ausfallen. Mixture Modelle, wo das Rauschen als Poisson Prozess simuliert ist, können solche Punkte erkennen.

- Präsentation der Resultaten. Kompakte und übergreifende Zusammenfassungen machen ein Clustering verwendbar. Viele Methoden haben die Eigenschaft mitintegriert: Subspace Clustering produziert eine DNF Zusammenfassung von jedem Cluster, Mixture Modelle generieren den Mittelpunkt und die Kovarianzmatrix für jedes Cluster. Dynamische Systeme erlauben eine Menge von Variablenwerten mit den größten Gewichten festzustellen.

- Gebrauch der Parameter. Alle Verfahren brauchen als Eingabe neben den Objekten, die zum Clustering ausgewählt wurden, einige Parameter. In der Praxis, ist es nicht leicht diese Parameter auszuwählen. Meistens, werden die Parameter durch einen Prozess des Resampling gesucht. Es wird berichtet, dass das Resampling die Suche nach den Parametern wesentlich erleichtert.

- Reihenfolge der Eingabe. Idealerweise funktionieren die Algorithmen unabhängig von der Reihenfolge der Eingabe der Objekte. Nicht in allen Verfahren ist das so (das hierarchische Sortieren).

10.3.2 Bezug zu anderen Themen

Innerhalb des KDD-Prozesses, der im Abschnitt 2 vorgestellt wurde, lässt sich Clustering-Lernverfahren in die Phase des Data-Mining einordnen.
Ein gutes Verständnis der vorliegenden Daten ist bei jedem Lehrverfahren unbedingt notwendig. Der Abschnitt 3 stellt eine Reihe von Werkzeugen, die auch beim Clustering sehr hilfreich sind, z.B. beim Mixture-Modelle Ansatz wird die statistische Expectation Maximization Methode zur Parameterschätzung benutzt.

Merkmalsgenerierung und -selektion sind Vorverarbeitungstechniken, die oft eine sehr wichtige Rolle für einen erfolgreichen Wissensentdeckungsprozess spielen, vor allem um längere Laufzeiten zu vermeiden (siehe Kapitel 12). Die Anzahl der Attributen wird beim Merkmalsselektion reduziert und damit erhöht sich die Effizienz der Clustering Verfahren. Beim begrifflichen Clustering wird die Technik der Merkmalsgenerierung und -selektion bei der Auswahl der begrifflichen Beschreibung der Cluster benutzt.
Verwandt mit der Aufgabe vom Clustering ist auch die Subgruppenentdeckung (siehe Kapitel 7). Die Subgruppenentdeckung hat zum Ziel, diejenigen Teilgruppen von dem gesamten Datenmenge zu identifizieren, die hinsichtlich eines bestimmten Merkmals gegenüber der Gesamtheit die statistischen Auffälligkeiten zeigen. Der Hauptunterschied vom Clustering zur Subgruppenentdeckung ist dass es hier keine Zielmerkmale gibt.
Die Stützvektormethode (Kapitel 5) wird heutzutage häufig beim Clustering eingesetzt [2]. Beim sogenannten Support Vector Clustering werden die Datenpunkte mittels des Gaussian-Kerns auf den mehrdimensionalen Attributenraum abgebildet, wo nach der minimale umschließende Sphäre gesucht wird. Diese Sphären, abgebildet auf den Datenraum zurück, bilden die gesuchten Cluster.

10.4 Zusammenfassung

Clustering ist die Zuordnung einer endlichen Menge von Objekte zu Gruppen, so dass die Objekte einer Gruppe sehr ähnlich zu einander sind. Die Objekte verschiedener Gruppen sind einander möglichst unähnlich. Konzeptuelles Clustering sucht außerdem eine kompakte und informative Beschreibung oder eine Zusammenfassung für jedes Cluster, die dieses Cluster möglichst gut definiert. Es gibt sehr Clustering-Methoden. Die Algorithmen unterscheiden sich in dem, wie sie die Objekte zu den Cluster zuordnen: erstens, partitionierend oder hierarchisch; zweitens, deterministisch oder mit Hilfe der Wahrscheinlichkeit; drittens, überlappend oder exklusiv usw. Neben dem traditionellen k-Clustering und hierarchischem Clustering Verfahren, wurde hier einige der neuen Methoden präsentiert: Mixture Modelle, Dynamische Systeme, Clique Graphs, Subspace Clustering und drei Varianten des hierarchischen Clustering.
Die wichtigsten Kriterien für die Wahl des Algorithmus für eine bestimmte Anwendung sind: die Daten und Cluster Gestalt und Typ, die Fähigkeit die Parameter richtig zu wählen und "Rauschen" zu behandeln. Beim Clustering ist das Benutzen vom Resampling sehr hilfreich.

Kapitel 11

Die Kombination mehrerer Modelle

Marc Twiehaus

Wenn kluge Menschen kritische Entscheidungen treffen müssen, holen sie in der Sache für gewöhnlich den Rat mehrerer Personen ein. Man wäre schlecht beraten, wenn man sich nur auf die Meinung einer einzelnen Person verlassen würde. Bagging und Boosting verfolgen diesen Ansatz und kombinieren die Vorhersagen mehrerer durch maschinelles Lernen erzeugter Modelle, um eine bessere Vorhersageleistung zu erzielen. Die unterschiedlichen Modelle werden dabei alle durch ein und denselben Algorithmus erzeugt, der bei jedem Durchlauf den zur Verfügung stehenden Trainingsdatensatz verändert und am Ende die einzelnen Vorhersagen durch eine Mehrheitsentscheidung zu einer einzigen Vorhersage zusammenfaßt. Obwohl Bagging und Boosting sich beide diesen Ansatzes bedienen, leiten sie die zugrunde liegenden Modelle auf unterschiedliche Art und Weise ab. Während beim Bagging keine Gewichtungen verwendet werden um die Veränderungen am Trainingsadtensatz vorzunehmen verwendet Boosting Gewichtungen um einerseits explizit nach Modellen zu suchen, die einander ergänzen, und anderseits um erfolgreicheren Modellen bei der Abstimmung mehr Einfluß zu geben. Beide können in den meisten Fällen eine größere Vorhersageleistung erbringen, als die einzelnen Modelle, und sind allgemeine Techniken, die sich auf numerische Vorhersageprobleme ebenso wie auf Klassifikationsaufgaben anwenden lassen. Grundvoraussetzung für beide Methoden ist nur, daß der Algorithmus zur Modellerzeugung instabil arbeitet, d.h. nur wenn eine Veränderung des zum Training verwendeten Datensatzes auch unterschiedliche Modelle erzeugt.

11.1 Bagging

Gegeben sei ein Trainingsdatensatz L bestehend aus Daten $\{(y_n, x_n), n = 1, ..., N\}$, wobei die y's entweder Klassenbezeichnungen oder numerische Werte angeben, und ein Algorithmus der ein Modell $\varphi(x, L)$ erzeugt, welches zu einem Input x eine Vorhersage y trifft. Angenommen man hätte mehrere Trainingsdatensätze L_k gleicher Größe, die willkürlich aus der Problemdomäne herausgegriffen wurden, zur Verfügung. Dann könnte man diese Sequenzen L_k nutzen um einen besseren Klassifizierer zu erhalten als $\varphi(x, L)$, der aus einem einzelnen Trainingsdatensatz konstruiert wurde. Wenn y ein numerischer Wert ist, ersetzt man dabei die Vorhersage von $\varphi(x, L)$ durch den Durchschnitt von $\varphi(x, L_k)$ über k, also durch $\varphi_A(x) = E_L\varphi(x, L)$, wobei E_L den Erwartungswert über L und φ_A den aggregierten Klassifizierer angibt. Wenn hingegen y eine Klassenbezeichnung $j \in \{1, ..., J\}$ repräsentiert, durch eine einfache Mehrheitsentscheidung, also $\varphi_A(x) = argmax_j N_j$, wobei $N_j = \sharp\{k; \varphi(x, L_k) = j\}$. Üblicherweise hat man aber nicht den Luxus eine ausreichend große Trainingsmenge zur Verfügung zu haben, um mehrere gleichgroße und unterschiedliche Trainingsdatensätze bilden zu können, und mehr Daten zu erhalten ist oft nicht möglich. Das Bagging umgeht dieses Problem durch das Bootstrap-Verfahren [23]. Hierbei werden die Trainingsdatensätze L_k durch unabhängiges Ziehen mit Zurücklegen aus der ursprünglichen Testmenge gezogen, so daß dann einzelne Instanzen in den L_k's auch doppelt oder gar nicht vorkommen können. Daher leitet sich auch der Name Bagging ab, als Bezeichnung für **B**ootstrap **agg**regat**ing**. Wie bereits erwähnt ist hierbei die Stabilität bzw. Instabilität des für die Konstruktion von φ verwendeten Algorithmus besonders wichtig. Wenn eine Veränderung im Trainingsdatensatz L, also ein Replikant von L, nur eine kleine Veränderung bei φ hervoruft, so wird der der aggregierte Klassifizierer φ_B keine bessere Leistung erzielen als der einfache Klassifizierer φ. Wenn allerdings kleine Veränderungen in L große Veränderung bei φ hervorrufen, kann das die Leistung erheblich verbessern. Beispiele für instabile Verfahren sind neuronale Netze, Klassifikations- und Regressionsbäume sowie die subset selection in linearer Regression, während z.B. k-nearest-neighbor Methoden stabil arbeiten.[8]

11.1.1 Algorithmus

Sei N die Anzahl der Instanzen im Trainingsdatensatz und t die Anzahl der Runden.

Wiederhole für t Runden:

- Ziehe zufälling und unabhängig mit Zurücklegen n Instanzen aus den Trainingsdaten

- Wende den Lernalgorithmus auf diese Stichprobe an

- Speichere das entstandene Modell

- Lasse jedes der t Modelle eine Vorhersage für eine neue unbekannte Instanz treffen und kombiniere diese Entscheidungen zu einer einzelnen Vorhersage:

  Bei numerischen Werten durch Durchschnittsbildung

  $$\varphi_{B(x)} = average_B \varphi(x, L^B),$$

  und bei Klassenbezeichnungen durch Mehrheitsentscheidung

  $$\varphi_B(x) = argmax_j N_j, \text{ wobei } N_j = \sharp\{k; \varphi(x, L_k) = j\}.$$

11.1.2 Analyse

Numerische Vorhersage

Sei jede Instanz $(y, x) \in L$ unabhängig mit gleicher Wahrscheinlichkeit aus der Verteilung P gezogen und sei $\varphi(x, L)$ der Klassifizierer. Dann sei der aggregierte Klassifizierer

$$\varphi_A(x, P) = E_L \varphi(x, L)$$

Seien Y, X zufällig aus der Verteilung P und unabhängig von L gezogen worden. Der durchschnittliche quadratische Vorhersagefehler (mean square error, MSE) e von $\varphi(x, L)$ ist dann:

$$e = E_L E_{Y,X}(Y - \varphi(x, L)))^2$$

Und der MSE e_a des aggregierten Klasifizierers φ_a

$$e_a = E_{Y,X}(Y - \varphi_a(X, P))^2$$

Sei x eine feste Eingabe- und y eine Ausgabevariable, dann gilt:

$$\begin{aligned} e_a &= y^2 - 2y\varphi_A(x, P) + \varphi_A^2(x, P) = (EZ)^2 && (11.1) \\ e &= y^2 - 2yE_L(\varphi(x, L)) + E_L(\varphi^2(x, L) = EZ^2 && (11.2) \end{aligned}$$

Wenn man nun die Gleichung $E_L\varphi(x, L) = \varphi_A$ auf den 2. Term und die Jensen Ungleichung $EZ^2 \geq (EZ)^2$ auf den 3. Term in (10.1) anwendet, ergibt sich:

$$E_L(y - \varphi_{e_a}(x, L))^2 \leq E_L\varphi_e^2(x, L)$$

Wenn man diesen Term nun über die gemeinsame x, y Ein-/Ausgabe-Verteilung integriert, ergibt sich, daß der MSE von $\varphi_A(x, L)$ kleiner als der MSE von φ(x,L) ist. Wieviel kleiner hängt von der Ungleichheit der Seiten in

$$[E_L\varphi(x, L)]^2 \leq E_L\varphi^2(x, L)$$

ab. Hier zeigt sich deutlich der Effekt der Instabilität. Wenn sich die $\varphi(x, L)$ nicht großartig bei Replikanten von L verändern sind beide Seiten beinahe gleich groß und die Aggregation bringt kaum Vorteile. Je variabler jedoch die einzelnen φ sind, desto mehr Verbesserung bringt die Aggregation.

Allerdings ist φ_A nicht nur vom Input x abhängig, sondern auch von der zugrundeliegenden Wahrscheinlichkeitsverteilung P aus der L gezogen wird, also ist die "gebaggte" Vorhersage eigentlich nicht $\varphi_A(x, P)$ sondern vielmehr

$$\varphi_B(x) = \varphi_A(x, P_L),$$

wobei P_L die Verteilung ist die sich mit Gewicht $\frac{1}{N}$ auf jeden Punkt $(x_n, y_n) \in L$ konzentriert, die sogenannte Bootstrap Approximation für P. Daher ist φ_B auch zwei Einflüssen ausgesetzt: Einerseits kann bei unstabilen Prozeduren eine Verbesserung durch die unschärfe der Jensen-Gleichung erreicht werden. Wenn anderseits die Prozedur stabil arbeitet, wird $\varphi_B = \varphi_A(x, P_L)$ nicht so genau auf Instanzen die aus P gezogen wurden arbeiten wie $\varphi_A(x, P) \simeq \varphi(x, L)$.

Es gibt also einen Übergang zwischen Instabilität und Stabilität, ab dem φ_B sich nicht verbessert und sich sogar verschlechtern kann, eine gute Illustration bietet das Experiment mit der linearen Regression mit subset-selction, das im nächsten Absatz vorgestellt wird. Außerdem sollte nicht vergessen werden, daß wenn $\varphi(x, L)$ bereits Nahe am Limit dessen liegt was überhaupt nur zu einem gegebenen Datensatz an Genauigkeit erreicht werden kann, das Bagging natürlich auch keine Verbesserung liefert. Auch hierfür gibt es ein Beispiel im Absatz 11.1.3 auf der nächsten Seite

Klassifikation

Der Effekt, mehrere Hypothesen zu kombinieren, kann auch vor dem Hintergrund eines theoretischen Überbaus betrachtet werden, der so genannten Bias-Varianz-Dekomposition[69]. Dabei handelt es sich um einen theoretischen Ansatz um festzustellen, wie gut die Menge der Testinstanzen zu einem Verfahren des maschinellen Lernens passt. Hierbei wird von der Annahme ausgegangen, daß eine unendliche Anzahl unabhängiger, gleichgroßer Trainingsmengen zur Verfügung steht um eine unendliche Zahl von Klassifizierern zu erzeugen. Diese verarbeiten dann alle eine Testinstanz und treffen per Mehrheitsentscheid eine einzelne Vorhersage. Selbst in dieser idealisierten Situation werden noch Fehler auftreten da kein Lernverfahren perfekt arbeitet und die Fehlerrate auch davon abhängt wie gut ein Verfahren des maschinellen Lernens für das vorliegende Problem geeignet ist. Die erwarte Fehlerrate wird berechnet, indem man den durchschnittlichen Fehler des kombinierten Klassifizierers über eine unendliche Anzahl unabhängiger gewählter Testinsatnzen ermittelt und wird als Bias des Algorithmus für das Lernproblem bezeichnet. Er misst also, wie gut die Lernmethode für das Problem geeignet ist, sozusagen den hartnäckigen Fehler, der selbst wenn eine unendliche Anzahl von Trainingsmengen für das Lernen zur Verfügung stehen würde, nicht eleminiert werden kann.
Eine zweite Fehlerquelle in einem erlernten Modell ergibt sich in der praktischen Situation aus der verwendeten Trainingsmenge, die ja unweigerlich endlich ist und von daher auch nicht 100% repräsentativ für die eigentliche Instanzpopulation sein kann. Der erwartete Wert dieser Fehlerkomponente über alle möglichen Trainingsmengen (der gegebenen Größe) und alle möglichen Testmengen wird als Varianz der Lernmethode bezogen auf das Problem bezeichnet. Der insgesamt zu erwartende Fehler eines Klassifizierers wird als Summe von Bias und Varianz berechnet, die Bias-Varianz-Dekomposition. Das Bagging, also die Kombination mehrer Klassifizierer senkt den Varianzfehler, und je mehr Klassifizierer beteiligt sind, desto größer wird dieser Effekt. Allerdings bedeutet dies nicht zwingend, daß auch der Gesamtfehler verringert wird. Bezüglich der Instabilität gilt aber auch hier das Gleiche wie bei der numerischen

Vorhersage.

11.1.3 Subset selection

Das Beispiel der subset selection bei linearer Regression verdeutlicht die Aussagen über die Instabilität. Hierbei handelt es sich um Daten der Form $L = \{(x_n, y_n), n = 1, ..N\}$, wobei $x = (x_1, ..., x_m)$ aus M Prediktoren besteht. Eine Möglichkeit der Vorhersage besteht darin $\varphi_1(x), ..., \varphi_M(x)$ Klassifizierer zu berechnen, wobei jedes φ_m linear in x ist und nur von m der M x-Variablen abhängig ist und nur eines der $\{\varphi_m\}$ als Klassifizierer gewählt wird. Eine übliche Methode zur Konstruktion der $\{\varphi_m\}$ besteht darin eine Vorwärtsauwahl der Variablen vorzunehmen. Es gibt auch ander Methoden, z.B. best subsets, eine Rückwärtsauswahl oder Varianten hiervon, aber sie alle sind instabil[8]. Für weitere Details zur subset selection verweise ich auf [9]. Breiman experimentierte mit einer simulierten Strukur mit M=30 Variablen und dem Modell:

$$y = \sum_m \beta_m x_m + \epsilon.$$

Dabei kam er zu folgenden Ergebnissen:
Im Falle von nur wenigen Koeffizienten ungleich Null ist die Prozedur schon fast optimal und man hat keine Verbesserung durch das Bagging. Dies demonstriert den offensichtlichen Effekt, daß Bagging nur Verbesserung bringt, wenn die ungebaggte Lernmethode nicht optimal arbeitet. Außerdem gibt es in Abhängikeit von der Anzahl der in die Prozedur aufgenommenen Variablen einen Punkt, ab dem das Bagging keine Verbesserung mehr bringt und sogar schlechter wird. Dieser Effekt liegt daran, daß die subset selection stabil ist, wenn man alle Variablen aufnimmt und mit sinkender Anzahl der Variablen instabiler wird[7].

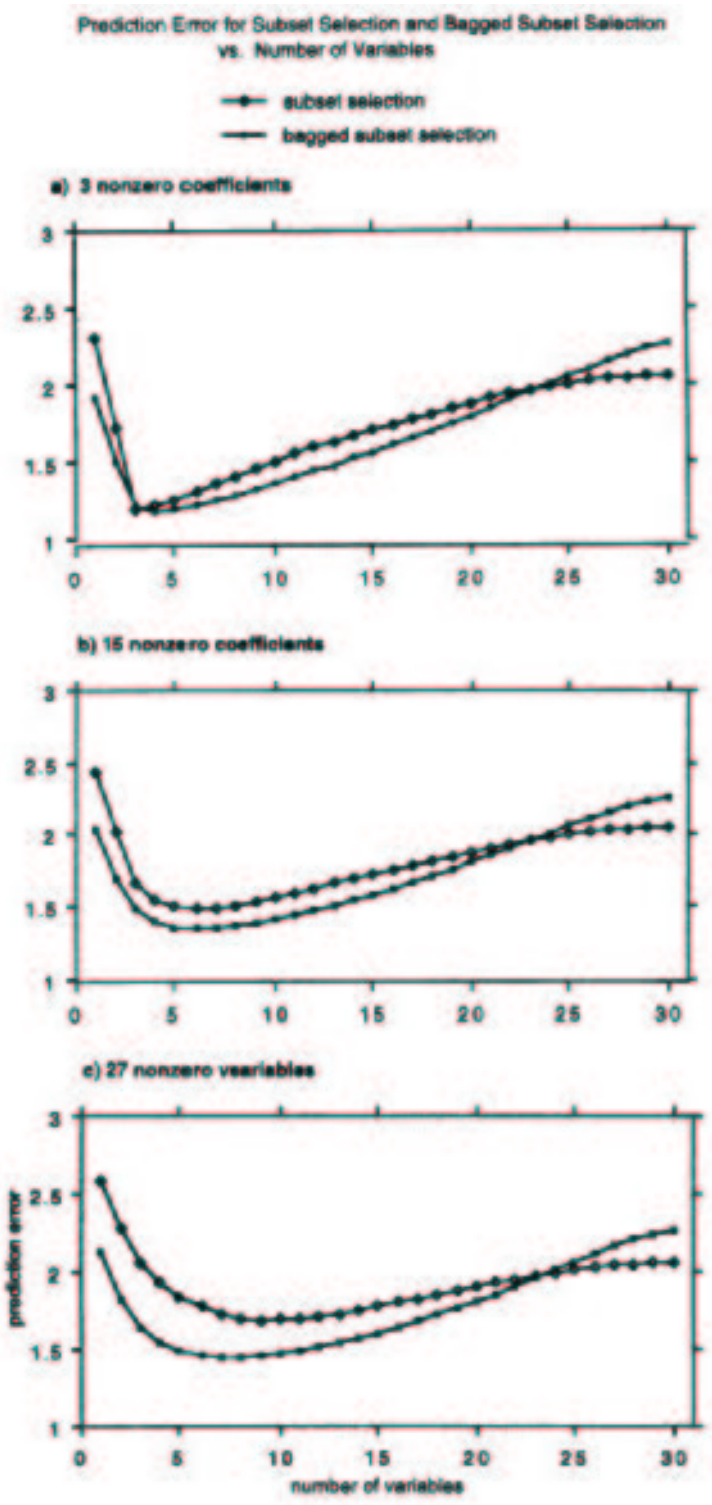


Abbildung 11.1: [7] subset selection vs. bagged subset selection

11.1.4 Anmerkungen

Was die Anzahl der Bootstrap Replikate betrifft, läßt sich sagen, daß im allgemeinen bei Reggressionsproblemen weniger benötigt werden als bei Klassifikationsproblemen, und die Anzahl mit der Anzahl der Klassen steigt. Breiman schlägt hier 25 für Reggresion und 50 für die Klassifikation vor. Die Größe der Replikate sollte dabei der Originalgröße des Trainigsdatensatztes entsprechen, Experimente mit größeren Replikaten führten zu keiner Verbesserung.
Abschließend läßt sich feststellen, daß das Bagging eine einfache Methode bietet die Genauigkeit zu verbessern und, da die Bildung der Replikate unabhängig erfolgt, sich hervorragend für eine Parallelverarbeitung eignet. Allerdings wird auch die Analyse der Ergebnisse erschwert, da man hierbei natürlich die Übersichtlichkeit eines einzelnen Baumes verliert.

11.2 Boosting

11.2.1 Vom Bagging zum Boosting

Das Boosting läßt sich am einfachsten als verfeinertes Bagging betrachten. Während beim Bagging zwar unterschiedliche Modelle erzeugt werden, nimmt der Algoritmus keinen gesteuerten Einfluß auf die Modellerzeugung. Daher können sich beim Bagging einzelne Modelle auch überlagern, sinnvoller wäre es allerdings Modelle zu haben, die sich nicht nur signifikant voneinander unterscheiden, sondern von denen jedes einen angemessen Prozentsatz der Daten korrekt behandelt. Idealerweise ergänzen sich diese Modelle dann und sind jeweils Spezialisten in einen Teil der Domäne, indem andere weniger leisten. Das Boosting nutzt nun diese Einsicht und versucht explizit solche Modelle zu erzeugen, indem Gewichtungen benutzt werden:

- Bei der Bildung der Replikate erhalten vorher falsch klassifizierte Instanzen ein höheres Gewicht bei der Bildung des nächsten Replikates und werden so ermutigt Experten für Instanzen zu werden, die von früheren Modellen falsch klassifiziert wurden.Im Gegensatz zum Bagging ist das Boosting also ein iterativer Prozeß.

- Bei der Mehrheitsentscheidung erhalten die Modelle ein Stimmgewicht, daß von der Leistung des Modells abhängt, wobei bessere Modelle natürlich ein höheres Gewicht erhalten und so mehr Einfluss auf die Entscheidung nehmen können.

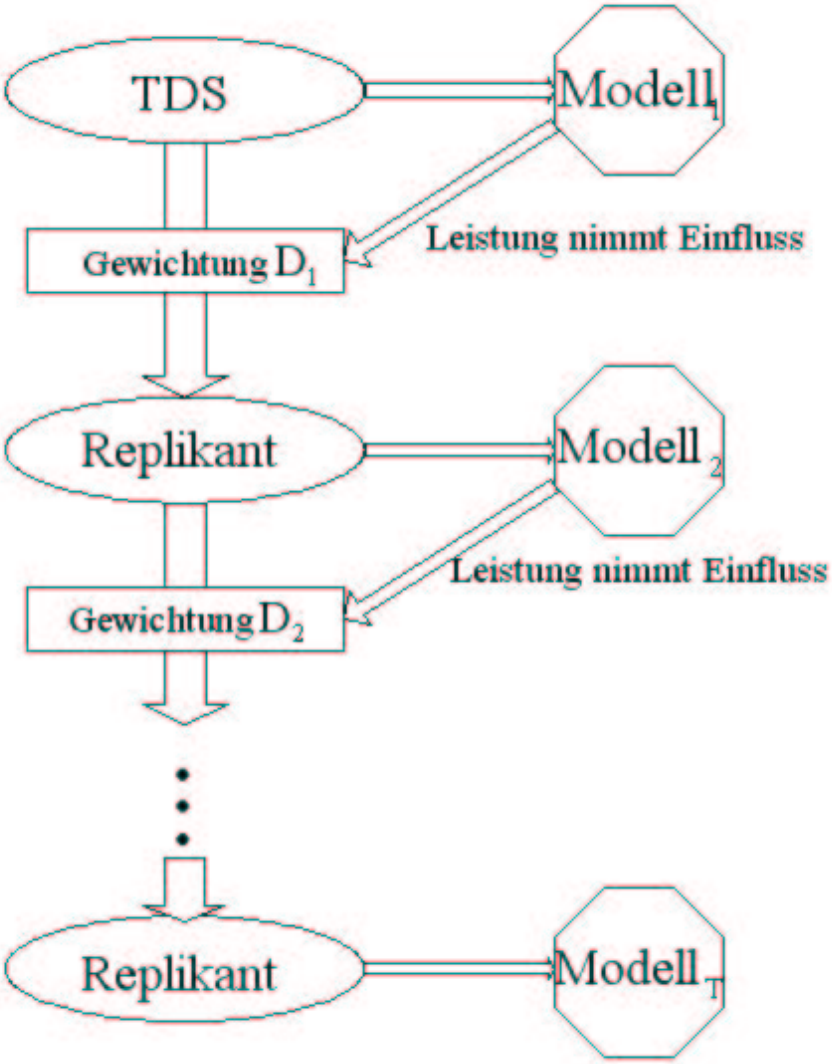


Abbildung 11.2: Der Boosting Prozess

11.2.2 Herkunft des Boosting

Während ihrer Arbeit an Valiants's PAC Lern-Modell waren Kearns und Valiant[44][45] die ersten die sich mit der Frage beschäftigten, ob man einen *schwachen Lerner*, der nur etwas besser als einfaches Raten ist, zu einem genaueren *starken Lerner* "boosten" kann. Beide Begriffe stammen aus eben diesem PAC-Modell das ich im Abschnitt 11.2.2 kurz vorstellen werde. Schapire[62] konnte dann 1989 den ersten beweisbaren polinomiellen Boosting Algorithmus vorstellen und ein Jahr später entwickelte Freund[29] einen wesentlich effizienteren Algorithmus, der obwohl in gewisser Hinsicht optimal, dennoch ebenso wie Schapire's Algorithmus einigen praktischen Nachteilen unterlag. Die ersten Experimente mit diesem frühen Boosting Algorithmus und Varianten davon wurden von Drucker, Schapire und Simard[22] an einer OCR-Aufgabe durchgeführt. Erst der Adaboost Algorithmus, der den ich hier vorstellen werde und der 1995 von Freund und Schapire[30] entwickelt wurde, löste viele der praktischen Schwierigkeiten denen die früheren Varianten des Boostings unterlagen[63].

PAC Modell

PAC-Lernen steht für probably approximatley correct learning, also annähernd korrektes lernen. Das PAC-Paradigma besagt, daß es völlig Ausichtslos ist, ein korrektes und vollständiges Lernverfahren zu fordern, das nach einer bestimmten Menge von Eingaben sicher und prompt das richtige Ergebnis abliefert und dann anhält[35]. Daher ist man dazu übergegangen seine Anforderungen an den Lerner abzumindern. Erstens verlangt man nicht, daß der Lerner komplett fehlerfrei arbeitet, sondern nur, daß sein Fehler durch eine Konstante ϵ beschränkt ist. Zweitens muss der Lerner nicht auf jeder Sequenz von Instanzen die aus den Trainingsdaten gezogen wurden erfolgreich sein, sondern verlangt ist nur, daß seine Wahrscheinlichkeit für ein Versagen auf einer Sequenz durch eine Konstante δ beschränkt ist. Der Abschwächung der Korrektheit stehen aber auch zwei Anforderungen gegenüber: Der Lerner soll in polynomiell beschränkter Rechenzeit zu einem Ergebnis kommen, nachdem er eine ebenfalls beschränkte Anzahl von Beispielen verarbeitet hat.

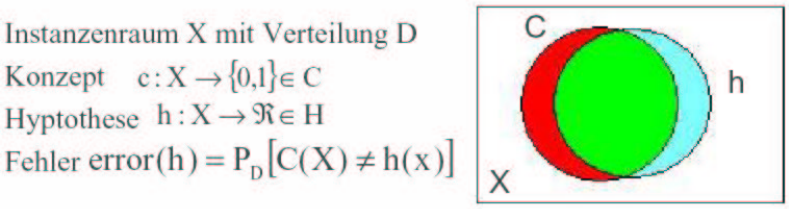


Abbildung 11.3: Das PAC-Modell

Definition: Gegeben ist eine Konzeptklasse C über einem Datensatz X der Länge N. Ein starker Lerner benutzt den Hypothesenraum H und kann C PAC-lernen, gdw.

- er für alle $c \in C$, alle festen aber unbekannten Verteilungen D und alle $\epsilon, \delta \in [0, \frac{1}{2}]$
- in einer Zeit, die durch ein Polynom über $\frac{1}{\epsilon}, \frac{1}{\delta}, |c|, |x|$ begrenzt ist
- mit einer Wahrscheinlichkeit von mindestens $1 - \delta$

eine Hypothese $h : X \to \Re \in H$ ausgibt, deren wahrer Fehler $error_D(h, c)$ nicht größer ist als ϵ.
Hierbei kann der Wert $Prob_D(h(x) \neq c(x))$ als Wahrscheinlichkeit für eine falsche Vorhersage für x betrachtet werden.

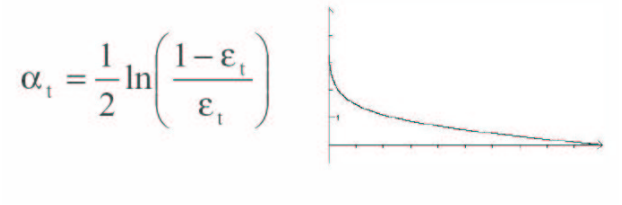
Ein schwacher Lerner erfüllt dieselben Bedingungen, jedoch fordert man für ihn nur das $\epsilon \leq \frac{1}{2} - \gamma$ ist, wobei γ eine Konstante ist oder sich poylnomiell verkleinert.

11.2.3 Algorithmus

Der Adaboost-Algorithmus:
Für eine vorgegebene Anzahl von T Iterationen, einen Trainingsdatensatz $L = (x_n, y_n), n = 1, ..., N$ mit $x_i \in X$ und $y \in Y = -1, +1$

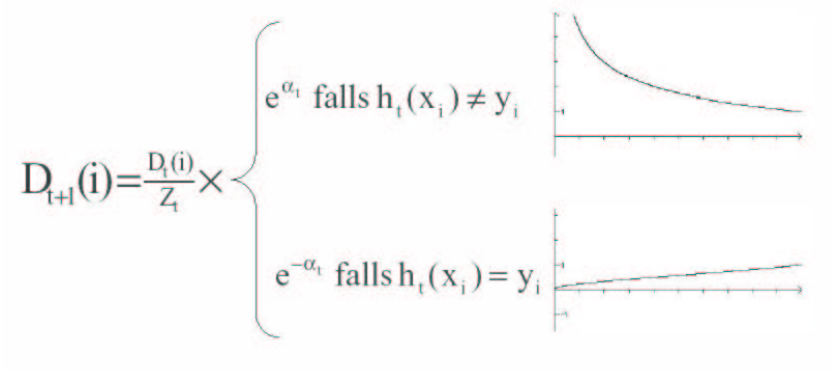
- Bilde das erste Replikat mit gleicher Ziehungswahrscheinlichkeit $D_1 = \frac{1}{N}$ für jede Instanz des Trainingsdatensatzes.
- Berechne die schwache Hyptothese $h_t : X \in \Re$ mit Fehler ϵ_t bezüglich D_t: $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$

- Berechne ein α_t in Abhängigkeit von ϵ_t:



  Wobei α so gewählt wurde um einerseits den Trainingsfehler schnell zu minimieren und den Grad der Bedeutung angibt, die Adaboost der zugehörigen Hypothese h_t zuordnet.

- Erhöhe bei der Bildung des nächsten Replikates die Ziehungswahrscheinlichkeiten für falsch klassifizierte Instanzen (daher der Name AdaBoost, für adaptives, also angepaßtes Boosten): Wobei Z_t ein Normalisierungs-



  faktor ist, damit D_{t+1} wieder eine Verteilung ergibt, also $\sum D_{t+1}(i) = 1$

- Ziehe in der nächsten Runde enstprechend dieser Verteilung und berechne h_{t+1}, ϵ_{t+1} und α_{t+1}

- Wiederhole dies für T Runden

- Kombiniere die nach T-Schritten erhaltenen Hypothesen $h_1, ..., h_T$ durch eine gewichtete Mehrheitsentscheidung und erhalte die endgültige Hypothese:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

11.2.4 Analyse

Wenn man Lernalgorithmen entwickelt und analysiert ist man neben der Laufzeit natürlich auch an dem wahren Fehler des Algorithmus interessiert. Für die folgende Analyse nehmen wir wiederum an, daß sowohl der Trainings- als auch der Testdatensatz unabhängig aus einer unbekannten aber festen Verteilung gezogen wurden und aus Gründen der Einfachheit, daß das Klassifizierungsproblem binär ist, also $Y \in -1, +1$. Freund und Schapir konnten beweisen, daß der Trainingsfehler des endgültigen geboosteten Klassifizierers in Abhängigkeit von seinem Trainingsfehler beschränkt ist durch die Größe m des Datensatzes, die VC-Dimension und die Anzahl der Durchläufe T. Genauer stellten sie sie fest, daß der Trainingsfehler mit hoher Wahrscheinlichkeit höchstens

$$\hat{Pr}[H(x) \neq y] + O\left(\sqrt{\frac{Td}{m}}\right)$$

beträgt. Diese Abschätzung deutet aber auf ein over-fitting bei zu vielen Durchläufen hin, also wenn T zu groß wird. Obwohl dies tatsächlich passieren kann, wurde empirisch nachgewiesen, daß boosting selten zur Überanpassung tendiert, selbst nach tausenden von Durchläufen. Weiterhin wurde festgestellt, daß Adaboost den wahren Fehler manchmal noch weiter verkleinert, obwohl der Trainingsfehler schon längst Null erreicht hatte. Da dies im Widerspruch zur obigen Abschätzung steht, kamen Schapir und andere[64] schließlich zu einer alternativen Analyse in Form der Margins der Trainingsdaten. Der Margin einer Instanz (x, y) ist dabei definiert als

$$margin(x, y) = \frac{y \sum_t \alpha_t h_t(x)}{\sum_t |\alpha_t|}$$

Der Margin ist eine Zahl im Bereich $[-1, +1]$ und nur positiv, wenn H die Instanz richtig klassifiziert. Zusätzlich gibt auch hier wieder die Größe des Wertes einen Vertrauensgrad für die getroffene Vorhersage an. Schapire und andere konnten zeigen, daß größere Margins bei den Testdaten in einer höheren obere Schranke für die Testdaten resultieren. Der wahre Fehler ist dabei höchstens:

$$\hat{Pr}[margin(x, y) \leq \theta] + O\left(\sqrt{\frac{d}{m\theta^2}}\right)$$

für beliebiges $\theta > 0$. Das interessanteste dabei ist, daß dieser Term völlig unabhängig von T, der Anzahl der Runden ist. Zusäztlich konnte gezeigt werden, daß Boosting sich besonders darauf konzentriert die Margins zu verbessern, weil es sich eben auf die Instanzen mit den kleinsten (positiven oder negativen) Margin-Werten konzentriert.

11.2.5 Anmerkungen

Es gibt eine Reihe von Varianten des Adaboost Algorithmus, z.b für Reggressions- und Mehrklassenprobleme, die hier nicht vorgestellt wurden, für Details verweise ich auf [63].
Das Boosting ist besonders für nicht zu komplexe Basisklassifizierer geeignet und wenn deren Traingsfehler nicht zu schnell zu groß werden (unter 50%). Zusätzlich muss man feststellen, daß Ausreißer in den Daten durch ihre Gewichtungen auch einen großen Einfluß auf das Ergebniss haben können. Dieser Nachteil wird allerdings durch den Umstand aufgehoben, daß sie aber auch gerade durch ihre hohe Gewichtung als solche identifiziert werden können. Das wichtigste beim Boosting ist allerdings, daß man schon einfache Basisklassifizierern, die auf den Replikanten der Trainingsdaten nur etwas besser als einfaches Raten arbeiten müßen, genügen um nur kleine Fehler auf den Trainings- und Testdaten zu erhalten.

11.3 Vergleich der Methoden

Bagging und Boosting sind beides allgemeine Verfahren, die sich auf eine Vielzahl unterschiedlicher Probleme anwenden lassen. Das Boosting ist dabei im allgemeinen genauer als das Bagging, aber seine Leistung ist dafür auch variabler als die des Bagging. Verallgemeinert gilt aber dennoch die Ungleichung Boosting > Bagging > einzelner Baum [31].

11.4 Einordnung in die Vortragsreihe

Zunächst möchte ich auf den Vortrag "Top-Down Induction of Decision Trees"4verweisen, die dort vorgestellten Entscheidungsbäume sind ein Beispiel für die Art von Modellen, die durch Bagging und Boosting kombiniert werden können. Die

ebenfalls dort erwähnte Entropie, die zum Aufbau eines einzelnen Baumes verwendet werden kann, illustriert gut die vorausgesetzte Instabilität. Es ist klar, daß sich die Entropie eines Attributes stark ändern kann, wenn bei einem Trainingsdatensatzreplikat Instanzen verdoppelt werden oder wegfallen. Dies bedeutet aber wiederum, daß die aus solchen Replikaten erstellten Modelle möglicherweise an einem bestimmten Knoten ein anderes Attribut wählen und sich so die Struktur des Baumes erheblich verändern kann.
Bagging und Boosting sind modellerzeugende Algorithmen, die im KDD-Prozess (siehe Vortrag "KDD-Prozess" 2) in der Phase des Data Minings benötigt werden. Auch wenn beide recht allgemeine Techniken darstellen, die sich auf eine Vielzahl von Problemen anwenden lassen, sollte man aber nicht vergessen, daß ihre Leistung zum einen auch stark von den zum Lernen verwendeten Daten abhängt. Im Vortrag "Merkmalsgenerierung" 12 gab es ein direktes Beispiel dafür und man sollte daher vor Anwendung der Algorithmen eine Merkmalsgenerierung in Betracht ziehen. Zum anderen gibt es natürlich auch Lernprobleme die sich ganz allgemein wenig für das Bagging oder Boosting eignen, nämlich solche bei denen es um eine Mustererkennung, die Findung von Assoziationsregeln oder Subgruppen geht. Hierbei geht es ja darum, die Problemdomäne in Klassen einzuteilen. Die Klassen die Algorithmen lernen könnten werden also durch diese Verfahren erst gefunden bzw. definiert. Beispiele für solche Probleme und die dafür geeigneten Methoden finden sich in den Vorträgen "Apriori"8, "Apriori für zeitliche Daten"9, "Clustering" 10 und "Subgruppenentdeckung"7. Die im Vortrag "Support Vector Machines" vorgestellten Methoden eigenen sich aufgrund der Stabilität von Stützvektormaschionen ebenfalls kaum für Bagging oder Boosting. Dennoch sind sowohl Bagging als auch Boosting wichtige Algorithmen, mit denen häufig versucht wurde eine Lösung der einzelnen KDD Cups (siehe Vortrag "Ausgewählte Aufgaben und Ergebnisse alter KDD Cups"13) zu finden. Der KDD Cup 1999 widmet sich sogar exklusiv diesem Thema, und die Gewinnerlösung bestand dabei aus einer Kombination von Bagging und Boosting.
Zum besseren Verständnis der Anaylse und Algorithmen verweise ich auf den Vortrag "Grundlagen der Statistik"3, hier finden sich nicht nur Details zu den Themen Verteilungen und Erwartungswerte sondern auch zur Abschätzung der Fehlerraten. Für die bei der Analyse von Adaboost verwendeten Begriffe der VC-Dimension und der Margins finden sich auch im Vortrag "Support Vector Machines"5 anschauliche Erläuterungen.

Kapitel 12

Merkmalsselektion und -generierung

Christian Kullmann

12.1 Einleitung

Ein allgemeines Problem aller intelligenten Agenten ist, auf welchen Teil der Aufgabe der Fokus gerichtet werden soll. Ein Agent, der ein Problem zu lösen hat, muss zuerst entscheiden, welche Aspekte des Problems relevant sind und welche nicht. Ein lernender Agent muss aus Erfahrungen lernen und dabei zwischen relevanten und irrelevanten Teilen unterscheiden können. Beim maschinellen Lernen wird einem Induktions-Algorithmus ein Set aus Trainingsdaten gegeben. Jede Instanz aus diesem Set wird beschrieben durch einen Vektor von Attributen (oder auch Merkmalen) und einer Klassen-Bezeichnung. Bei der medizinischen Diagnose zum Beispiel, besteht jede Instanz aus Attributen wie Alter, Gewicht und Blutdruck und als Klassenbezeichner kann angegeben sein, ob der behandelnde Arzt eine Herzkrankheit diagnostiziert hat, oder nicht.
Die Aufgabe des Induktions-Algorithmus besteht nun also darin, anhand der gegebenen Daten zu lernen, wovon die Klassifizierung abhängt um so zukünftige Daten selber klassifizieren zu können.
Beim Problem der Merkmalsselektion beschäftigt man sich nun mit der Frage,

wie man die für unsere Klassifizierung relevanten Merkmale ermitteln kann, so daß der Algorithmus sich voll diesen Merkmalen widmen kann, und den Rest ignoriert. Diesem Problem widmen wir uns direkt im kommenden Abschnitt.
Eventuell reichen die bereits vorhandenen Attribute aber nicht aus, eine Klassifizierung effizient durchzuführen, weil einige der "primitiven" Merkmale noch miteinander verknüpft werden müssen. Dieses Problem gehört in den Bereich der Merkmalsgenerierung. Hier geht es um die Frage, auf welche Art und mit welchen Operatoren man die optimalen Merkmale findet. Einige Antwortmöglichkeiten finden sich im Abschnitt 3.
Im Anschluss daran geht es darum, wie man die beiden Ansätze verknüpft, da sich die Frage schon fast aufdrängt, warum man denn nicht beides mache. Antworten auf diese Fragen sind in Abschnitt 4 zu finden.
Schließlich wird in Abschnitt 5 versucht, eine Verbindung zu den anderen in dieser Arbeit angesprochenen Themen zu schaffen und in Abschnitt 6 werde ich noch einige abschliessende, zusammenfassende Bemerkungen machen.

12.2 Merkmalsselektion

12.2.1 Überblick

Wenn es darum geht, einen Algorithmus zu schreiben, oder einen bereits vorhandenen Algorithmus eine bestimmte Aufgabe erledigen zu lassen, stehen oft Fragen im Raum wie "Wie lange braucht dieser Algorithmus für die gestellte Aufgabe?", "Wieviele Ressourcen wird er für die Aufgabe benötigen?" oder "Wie gut wird er die Aufgabe lösen?".
Die ersten beiden Fragen stellen zwar in der Komplexitätstheorie und beim Entwurf von Algorithmen eine essentielle Frage dar, jedoch soll der Fokus hier darauf liegen, welche Vorkehrungen ich treffen kann, um ein optimales Ergebnis bei einer Lernaufgabe zu erzielen.
Hier wird einem Algorithmus ein Set an Daten gegeben, anhand derer er lernen soll, zu erkennen, welche Merkmale der Instanzen Einfluss auf ihre Klassifizierung haben. Beim Problem der Merkmalsselektion geht es vor allem darum, bei den Merkmalen die für unsere Klassifizierung relevanten von den irrelevanten Merkmalen zu trennen und unseren Algorithmus also nur mit den relevanten Merkmalen zu "füttern".
Im günstigsten Fall hat man einen Experten zur Hand, der einem genau sagen kann, welche Attribute für die Klassifizierung ausschlaggebend sind, und welche nicht. Doch leider ist in den meisten Fällen ein solcher Experte eben nicht verfügbar, so daß man die Merkmalsauswahl auf einem anderen Wege

durchführen muss.
Es ist oft schon deswegen schlecht, alle Merkmale in Betracht zu ziehen, weil manche Attribute das Ergebnis verfälschen können, bzw. den Algorithmus derart in die Irre führen können, dass er ein eigentlich irrelevantes Merkmal für sehr relevant hält. Zum Beispiel würde ein Entscheidungsbaum-Algorithmus über medizinischen Daten fast immer die Sozialversicherungsnummer der Patienten als ein relevantes, wenn nicht sogar als das einzig relevante, Merkmal identifizieren und versuchen, zukünftige Klassifzierung anhand dieses Merkmals vorzunehmen. Der Grund hierfür ist in der Beschaffenheit der Algorithmen zu suchen. (siehe hierzu das Kapitel über Entscheidungsbäume). Unsere eigentliche Intention sollte also sein, die optimale Auswahl an Merkmalen zu finden, so dass die Genauigkeit der klassifizierenden Funktion einer Aufgabe maximiert wird.
Doch wie gehen wir nun an eine Merkmals-Auswahl, wenn wir kaum Ahnung von der Bedeutung der Merkmale für die Aufgabe haben?
Hierfür gibt es zwei Vorgehensweisen, die im folgenden genauer erklärt werden.

12.2.2 Filter

Überblick

Der Filter geht von der Annahme aus, dass es besser ist, eine Merkmalsauswahl zu treffen, bevor der eigentliche Lernalgorithmus mit den Merkmalen konfrontiert wird. Dazu werden sämtliche Merkmale vorher schon betrachtet, und, je nach Filter-Algorithmus, entsprechend einer bestimmten Güte ausgewählt.
Im einfachsten Fall wählt der Algorithmus sämtliche Merkmale aus, aber das kann, wie bereits im vorangegangenen Abschnitt beschrieben, nicht unsere Intention sein.
Eine Möglichkeit besteht darin, die Merkmale im Hinblick darauf zu gewichten, wie stark sie auf die Klassifizierung einwirken. Letztlich werden dann alle die Merkmale ausgewählt, deren Gewichts-Wert einen bestimmten, vom Nutzer vorgegebenen Wert überschreitet.
Der grösste Nachteil des Filter-Ansatzes ist ganz klar, dass er den Lernalgorithmus völlig ausser Acht lässt, und auch kein Feedback mehr von diesem bekommt. Also kann man nicht wirklich herausfinden, wie gut die Auswahl an Merkmalen letztlich war, bevor der Lernalgorithmus dann ein Ergebnis abgeliefert hat.

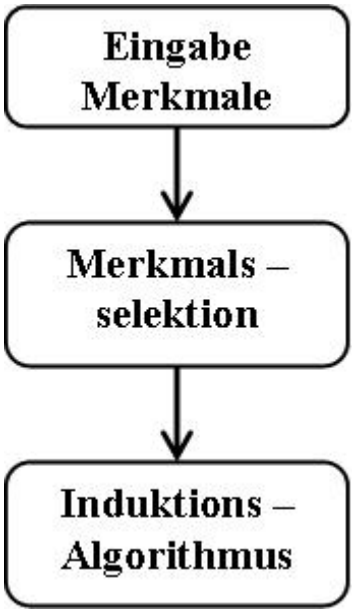


Abbildung 12.1: Der Filter-Ansatz, in dem die Merkmale unabhängig vom Induktions-Algorithmus gefiltert werden.

Filter - Ansätze

FOCUS ist ein Merkmalsselektions-Algorithmus für rauschfreie bool'sche Domänen. Er betrachtet alle Selektionen von Merkmalen, um dann diejenige auszuwählen, die minimal ist und ausreicht, die Klassifizierung aller Instanzen des Trainingssets vorzunehmen.
Ein grosser, bekannter Nachteil von FOCUS ist z.B., daß er bei einer medizinischen Diagnose-Klassifizierung die Sozialversicherungsnummer als einziges Merkmal auswählt, sofern dieses vorhanden ist.
RELIEF ist ein Algorithmus, welcher jedem Merkmal Relevanz-Gewichte zuweist, welche die Relevanz des jeweiligen Merkmals auf den zu klassifizierenden Wert angeben sollen. Der Algorithmus versucht alle relevanten Merkmale zu finden, indem er zufällig Instanzen aus dem Trainingsset auswählt, und die Merkmale in diesen Instanzen auf ihre Korrelation mit dem Zielmerkmal überprüft. Anhand dieser Überprüfung werden dann die jeweiligen Gewichte zugeordnet. Das Problem hierbei ist, daß in realen Domänen viele Merkmale mit dem Klassen-Merkmal korrelieren, wobei allerdings viele davon nur schwach relevant sind, und daher auch nicht von RELIEF aussortiert werden.
Baum-Filter verwenden einen Entscheidungs-Baum-Algorithmus, um eine Auswahl an Merkmalen zu treffen, typischerweise für einen Nächster-Nachbar-Algorithmus. Das kann für einige Datensätze ganz gut funktionieren, jedoch

kann es auch zu einer schlechten Auswahl an Merkmalen kommen, da nicht alle Auswahlen, die gut für Entscheidungs-Bäume funktionieren auch gut für Nächste-Nachbarn-Algorithmen sind.

12.2.3 Wrapper

Überblick

Beim Wrapper-Ansatz wird im Gegensatz zu der Filter-Variante der Induktions-Algorithmus nicht aussen vor gelassen. Man verwendet ihn als Teil der Suche nach einer optimalen Merkmalsauswahl. Hierzu wird mittels eines Greedy- oder Best-First-Search-Algorithmus der Suchraum aller Merkmalsauswahlen durchsucht, welche dann evaluiert werden mittels des Induktions-Algorithmus. Dieser stellt anhand der Auswahlen Hypothesen auf, welche dann wieder dazu dienen, die Auswahlen zu evaluieren und gegebenenfalls zu verwerfen oder behalten. Wichtig ist, dass nach Möglichkeit die Beispiel-Daten in ein Trainingsset und ein Testset unterteilt werden, so daß der Algorithmus versucht anhand des Trainingssets zu lernen und seine Hypothese dann mittels der Testdaten zu überprüfen. Letztlich liefert dieser Mechanismus dann eine Merkmals-Auswahl welche sich schon mit dem Algorithmus, dem sie jetzt zur Verfügung gestellt wird für die eigentliche Klassifizierungsaufgabe, bewährt hat.

Um die entsprechenden Auswahlen an Merkmalen zu finden, stellen wir uns die Auswahlen von Merkmalen als Suchraum dar, in dem jeder Knoten n Bits enthält, die die einzelnen Merkmale angeben, und die 0 sind, wenn das Merkmal in der Auswahl nicht vorhanden ist, und 1, wenn es vorhanden ist. Operatoren stellen die Verbindung zwischen den Knoten her, so daß ein Operator immer nur angibt, daß **ein** Merkmal hinzugenommen oder entfernt wurde. In Abbildung 12.3 ist ein solcher Suchraum für ein Problem mit vier Merkmalen dargestellt. Die Größe eines solchen Suchraums mit n Merkmalen beträgt $O(2^n)$.

Unser Ziel ist es nun, den Zustand mit der besten Bewertung zu finden, wobei wir dafür eine Heuristik verwenden, um unsere Suche zu lenken. Da wir nichts über die tatsächliche Genauigkeit des induzierten Klassifizierers wissen, verwenden wir die geschätzte Genauigkeit sowohl als Heuristik als auch als Bewertungsfunktion. Für die Bewertung verwenden wir eine fünf-fache Kreuzvalidierung, welche wir mehrfach wiederholen.

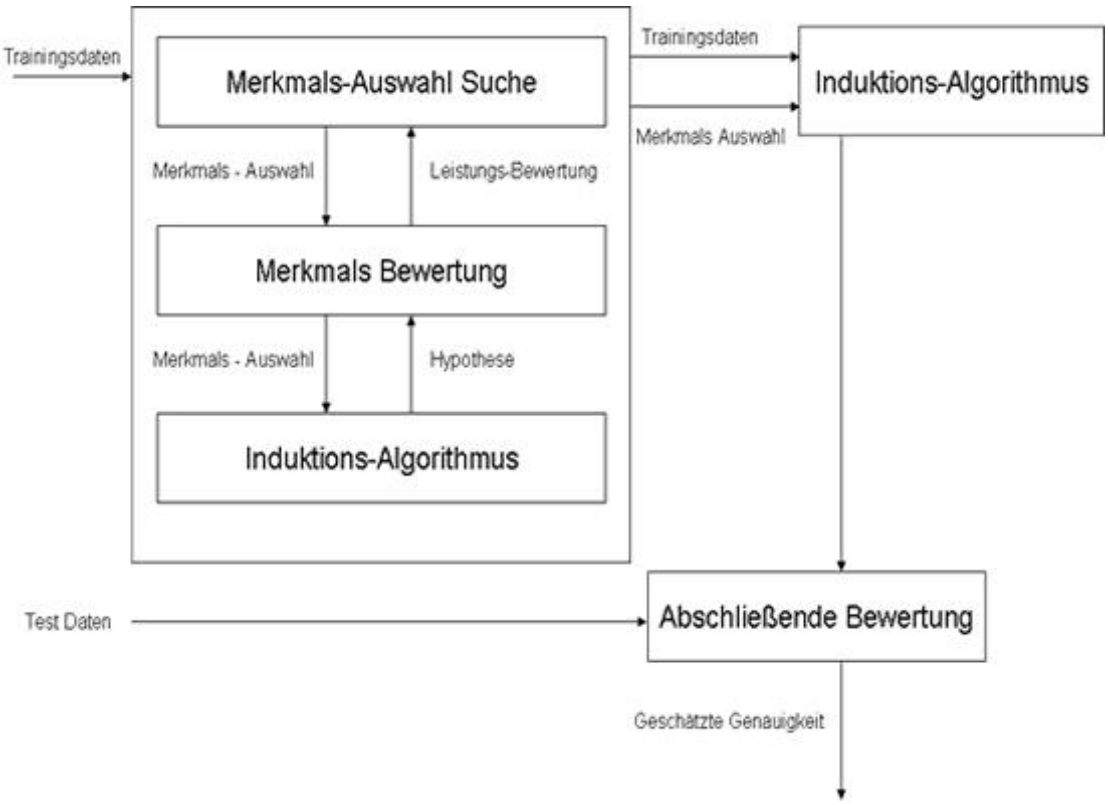


Abbildung 12.2: Der Wrapper-Ansatz, bei dem der Induktions-Algorithmus selbst der Mechanismus ist, anhand dessen die Merkmale ausgewählt werden.

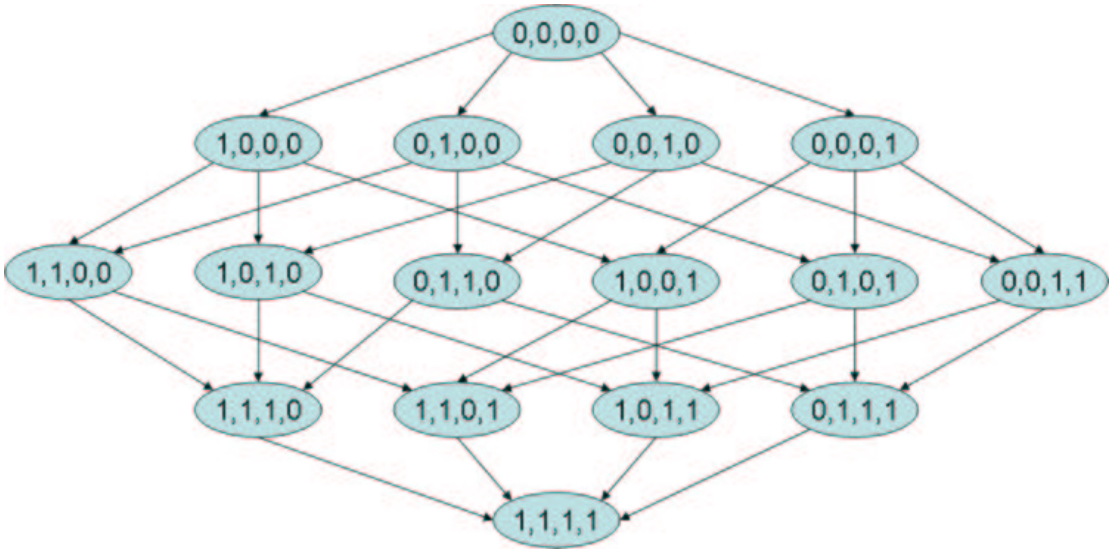


Abbildung 12.3: Jeder Knoten ist mit allen Knoten verbunden, die sich von ihm dadurch unterscheiden, daß ein Merkmal addiert oder entfernt wurde

Relevanz

Doch wonach richtet es sich, welche Merkmale gut geeignet sind, also relevant, und welche nicht?
Als Erklärung für die Relevanz zieht man den Zusammenhang mit einer Bayes'schen Regel heran:

- Ein Merkmal X ist **stark relevant**, wenn das alleinige Entfernen von X eine optimale Bayes'sche Regel so einschränkt, daß der Leistungsabfall signifikant ist.
- Ein Merkmal X ist **schwach relevant**, wenn es nicht stark relevant ist, aber in einigen Fällen die Vorhersage-Genauigkeit einer optimalen Bayes'schen Regel steigern kann.
- Ein Merkmal ist **irrelevant**, wenn es weder stark noch schwach relevant ist.

Wie sieht jetzt aber die genaue (mathematische) Definition von Relevanz aus?

Starke Relevanz
Sei S eine Auswahl aller Merkmale, und sei $S_i = S - \{X_i\}$. Ein Merkmal X_i

ist stark relevant, falls Werte x_i, y, und s_i existieren, für die gilt:
$p(X_i = x_i, S_i = s_i) > 0$ so dass

$$p(Y = y|X_i = x_i, S_i = s_i) \neq p(Y = y|S_i = s_i).$$

Schwache Relevanz
Ein Merkmal X_i ist schwach relevant, wenn es nicht stark relevant ist und eine Teilauswahl an Merkmalen S_i' aus S_i existiert, für die einige x_i, y und s_i' existieren mit $p(X_i = x_i, S_i' = s_i') > 0$ so daß

$$p(Y = y|X_i = x_i, S_i' = s_i') \neq p(Y = y|S_i' = s_i').$$

Best-First-Search

Best-First-Search ist eine solide Methode, welche nach der Idee vorgeht, die vielversprechendsten Knoten, die wir bis jetzt haben und die noch nicht expandiert wurden, zu expandieren. Um zu verhindern, daß der Such-Algorithmus in lokalen Maxima terminiert, lässt man ihn noch mindestens k Knoten weiter untersuchen, wenn er eine scheinbar optimale Lösung gefunden hat, und lässt ihn erst dann terminieren, wenn auch dann keine bessere Lösung gefunden wurde.

Der Suchraum: Zusammengesetzte Operatoren

Wie wir bereits feststellen mussten, ist der Suchraum sehr gross. Um unsere Sucharbeit zu erleichtern und den ganzen Mechanismus ein wenig zu beschleunigen, können wir den Suchraum dynamisch so modifizieren, daß wir uns schneller durcharbeiten können. Aber wie stellt man sowas an?
Nun der Suchraum ist für gewöhnlich so aufgebaut, daß ein Knoten eine Auswahl an Merkmalen darstellt, und eine Kante einen Operator repräsentiert, welcher von einem Knoten zum nächsten ein Merkmal hinzufügt oder abzieht. Das Hauptproblem ist jetzt, daß die Suche jeden Knoten auf dem Pfad von der Anfangs-Auswahl bis zur besten Auswahl expandieren und evaluieren muss. Man kann sich aber auch überlegen, daß vielleicht eine Mischung aus zwei der evaluierten Kinder eines Knotes, die besonders gut abgeschnitten haben bei

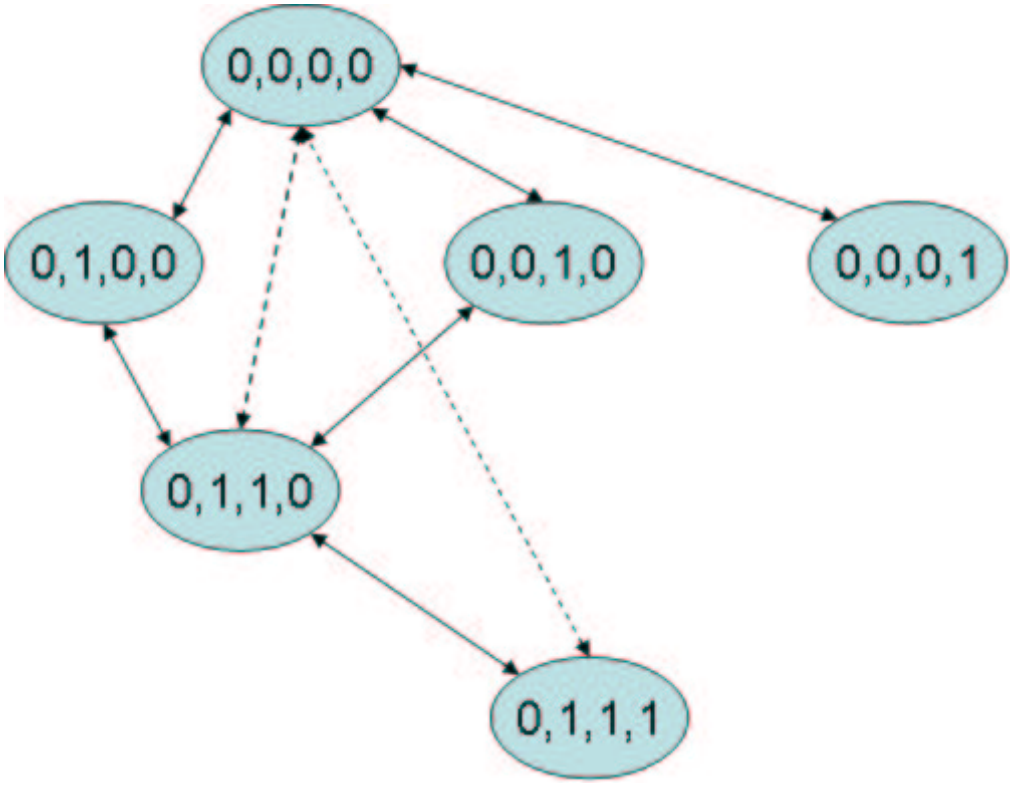


Abbildung 12.4: Die gestrichelten Linien stellen zusammengesetzte Operatoren dar

der Bewertung, noch besser abschneiden wird. Also verknüpfen wir die beiden Operatoren, welche zu diesen Kindern geführt haben und schon bewegen wir uns wesentlich schneller durch unseren Suchraum als vorher. Im nächsten Schritt könnten wir dann den nächstbesten der ursprünglichen Kinder hinzunehmen, und den Operator mit dem gerade erstellten zusammengesetzten Operator verknüpfen, um weiterzukommen.
Ein Beispiel hierfür ist in Abbildung 12.4 zu sehen.

12.3 Merkmalsgenerierung

12.3.1 Überblick

Im letzten Kapitel ging es um die Auswahl von Merkmalen für eine Lernaufgabe. Was aber, wenn die Merkmale, die gegeben sind, keine ausreichende

Beschreibung liefern, um eine Klassifizierung vorzunehmen? In diesem Falle müssen wir versuchen, aus den bereits bestehenden, sog. *primitiven* Merkmalen neue Merkmale zu generieren, die möglicherweise die Klasse unserer Instanzen besser bestimmen können. Hierfür gibt es eine Definition von Metheus und Rendell:

> [Constructive Induction is] the application of a set of constructive operators to a set of existing features resulting in the construction of one or more new features intended for use in describing the target concept.

Als Beispiel sei hier erneut auf eine medizinische Diagnose hingewiesen:
Für gewöhnlich sind in medizinischen Daten nur das Geburtsdatum und noch das Behandlungsdatum angegeben. Obwohl jetzt ein Mensch aus dem Geburtsdatum schnell das Alter des jeweiligen Patienten im Kopf errechnen kann, stellt das Geburtsdatum nur ein weiteres Datum für einen Algorithmus dar. Wenn wir jetzt aber ein neues Merkmal *Alter* generieren, indem wir das Geburtsdatum vom Behandlungsdatum abziehen, so haben wir ein neues, aussagekräftiges Merkmal gewonnen.
Man kann Merkmals-Generierungs-Systeme in vier Klassen unterteilen:

- *Daten gesteuerte Generierungs-Systeme* die die Eingabe-Daten analysieren, speziell die Beschreibungen in den Besipielen, um dann darauf basierend Änderungs-Vorschläge für die Repräsentation zu machen.
- *Hypothesen gesteuerte Generierungs-Systeme*, welche den Repräsentations-Raum Stück für Stück transformieren, indem sie die induzierte Hypothese analysieren, die in einem Schritt erzeugt wurde, um die dort entdeckten Muster in den Merkmalen für den nächsten Schritt zu verwenden.
- *Wissensgesteuerte Genereriungs-Systeme*, welche Experten-Wissen der betrachteten Domäne heranziehen, um Merkmale zu generieren und/oder zu verifizieren.
- *Kombination der oben genannten Systeme*, welche einfach verschiedene Systeme verwenden, um neue Merkmale zu generieren, mit denen dann der Repräsentationsraum verändert wird.

Doch was für Operatoren kann man nun einsetzen, um neue Merkmale zu erzeugen?

Hier stehen uns eine ganze Reihe an Operatoren zur Verfügung, von denen die einfachsten und am häufigsten verwendeten die Konjunktion, Disjunktion und Negation sind.
Außerdem wird häufig von den sog. *M-von-N-Operatoren* Gebrauch gemacht. Diese gibt es in diversen Varianten, nämlich:

- Mindestens M-von-N: Dieser Operator besteht aus der ganzen Zahl M und N Konditionen basierend auf bereits existierenden Attributen. Der Operator nimmt den Wert **wahr** an, wenn mindesten M von N Konditionen **wahr** sind.
- Höchstens M-von-N: Analog zu *min. M-von-N*, jedoch dürfen **höchstens** M von N Konditionen **wahr** sein.
- Exakt M-von-N: Analog zu *min. M-von-N*, jedoch müssen **genau** M von N Konditionen **wahr** sein.

Desweiteren existiert noch ein X-von-N Operator, der ähnlich wie die M-von-N Operatoren funktioniert, jedoch die ganze Zahl X zurückliefert, die angibt, dass X von N Konditionen **wahr** sind.

12.3.2 Lineare Gleichungen als Operator

In ihrem Artikel [33] beschreiben Gama und Brazdil eine weitere Möglichkeit, mit der man in Domänen, die zumindest teilweise aus numerischen Merkmalen bestehen, arbeiten kann, sofern man vor dem Problem steht, einen Entscheidungsbaum verwenden zu müssen. Dass dieser sicher nicht unbedingt die beste Wahl als Lernalgorithmus für diese Umgebung ist, sollte uns nicht davor zurückschrecken lassen, den Ansatz weiter zu verfolgen, da er uns eine weitere Möglichkeit für andere Probleme eröffnet. Doch zuerst das Problem:
Es sei ein künstliches Zwei-Klassen-Problem gegeben, welches durch zwei numerische Attribute definiert ist. Nachdem ein C4.5 (siehe hierzu das Kapitel Entscheidungsbäume) sich mit dem Problem beschäftigt hat, erhalten wir einen Entscheidungsbaum mit 77 Knoten.
Bei näherer Betrachtung der Pfade des Knotens finden wir heraus, daß systematisch dieselben Attribute immer wieder geprüft werden. Zum Beispiel kann ein Pfad folgendermassen aussehen:
IF $at_1 \leq 0.493$ AND $at_2 \leq 0.32$ AND $at_1 \leq 0.247$ AND $at_2 \leq 0.196$ AND $at_1 \leq 0.141$ AND $at_2 > 0.116$ THEN Class -
Das bedeutet, daß sich der Baum an eine Abschätzung annähert, indem er eine

Intervall-Schachtelung simuliert. Wäre es nicht vielleicht einfach gewesen, die beiden Attribute direkt miteinander zu vergleichen?
Wenn wir ein neues Attribut, at_3, erzeugen, können wir hier vielleicht einiges vereinfachen. Für einen Vergleich stehen uns zwei Möglichkeiten offen:

- $at_3 = at_1 - at_2$
- $at_3 = at_1/at_2$

Die daraus resultierenden Bäume würden folgender Gesetzmäßigkeit folgen, hätten also jeweils nur 2 Ebenen:

$at_3 = at_1 - at_2$	$at_3 = at_1/at_2$
IF $at_3 \leq 0$ THEN Class –	IF $at_3 \leq 1$ THEN Class –
IF $at_3 > 0$ THEN Class +	IF $at_3 > 1$ THEN Class 1

Wir können jedoch das ganze noch ein wenig optimieren, und für zukünftige Probleme handhabbar machen, indem wir lineare Gleichungen verwenden. Anstatt nun einfach eine einfache Division bzw. Subtraktion zu verwenden, erzeugen wir die Formel $H = 0 + 12 * at_1 - 12 * at_2$, wobei H eine Hyper-Ebene darstellt, welche durch diese Gleichung erzeugt wird. Das neue Attribut (at_3) wird erzeugt, indem nun die Beispiele über diese Hyper-Ebene projiziert werden. Lässt man nun C4.5 erneut über die Beispiele laufen, muss lediglich ein einziger Test gemacht werden, nämlich

$$\text{IF } at_3 \leq 0 \text{ THEN Class} - \text{ ELSE Class} +$$

Wendet man nun diese Operatoren auf ein Problem an, wie es [33] getan haben, dann sollte man nicht für sämtliche Attribute sofort riesige lineare Gleichungen erzeugen, und dann versuchen, daraus die richtige zu finden. Gama und Brazdil haben gezeigt, daß sich bei Anwendung dieses Operators eine sukzessive Erzeugung der Gleichungen und deren Implentierung Top-Down im Baum lohnt. Die Attribute werden erst an jedem Entscheidungspunkt erzeugt, nicht bereits zu Beginn der Untersuchung.
Abschliessend zu diesem Operator sei jedoch noch einmal darauf hingewiesen, daß es sich dabei nicht um das "Non-Plus-Ultra" der Operatoren handelt, sondern lediglich um eine Hilfe, wenn man zwingend einen Entscheidungsbaum - Algorithmus in einer zumindest teilweise numerischen Domäne anwenden

möchte oder muss. Es existieren noch viele weitere Operatoren, und es sei weiterhin darauf hingewiesen, daß es sicherlich auch noch Operatoren gibt, an deren Anwendung bisher niemand dachte.

12.4 Kombination von Selektion und Generierung

12.4.1 Überblick

Kommen wir nun zu der Kombination der beiden betrachteten Verfahren: der Selektion und der Generierung von Merkmalen. Die Selektion kann uns eine gut geeignete Auswahl der vorhandenen Merkmale liefern, nachdem was wir bis jetzt wissen. Stellen wir fest, daß die vorhandenen Merkmale nicht ausreichen, dann konstruieren wir uns eben welche. Doch treten die beiden Lösungen selten unabhängig voneinander auf. So kann man sicherlich nicht alle Merkmale, die man sich mühselig konstruiert hat, für eine Klassifizierung gebrauchen. Und wenn man nur selektieren möchte, dann steht man oft vor dem Problem, welches die Existenz der Merkmals-Generierung rechtfertigt. Doch kann man nun die beiden Bereiche einach so kombinieren?
Ja und nein. Man kann bzw. sollte nicht beides gleichzeitig anwenden, sondern muss beides zeitlich versetzt zueinander anwenden. So können dann aus den selektierten Merkmalen neue konstruiert werden, und daraus dann wieder einige selektiert werden usw...
Aber es gibt noch ein weiteres Problem: Wie wir bereits bei der Selektion feststellen mussten, ist der Suchram der Merkmale sehr gross, und kann sich durch Anwendung der Generierung noch weiter vergrößern. Die Kombination könnte uns also vor unlösbare Probleme stellen.

12.4.2 Genetische Algorithmen

Ein möglicher Lösungsansatz für das beschriebene Problem sind genetische Algorithmen. Sie scheinen sich aus mehreren Gründen anzubieten, da

- sie grosse Suchräume effizient durchsuchen können,

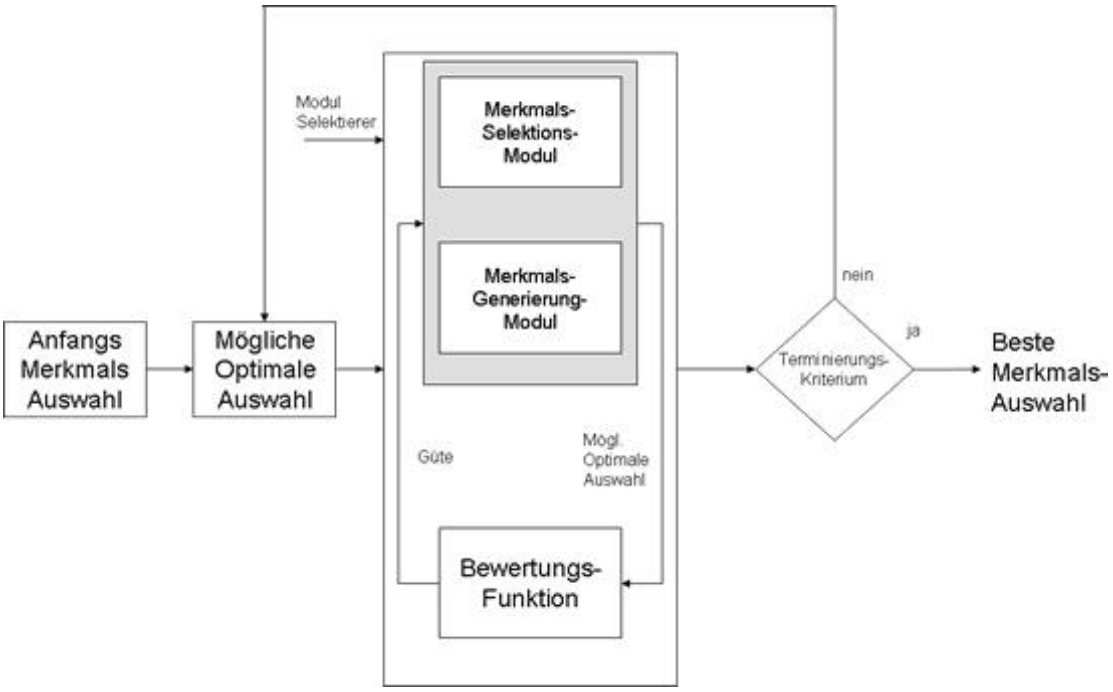


Abbildung 12.5: Eine Darstellung des Systems unter Verwendung eines genetischen Algorithmus.

- und sie sehr robust sind auf einer grossen Bandbreite von Problem-Domänen

$\Longrightarrow$ Sie scheinen also die optimale Wahl zu sein! Selbstverständlich sind genetische Algorithmen dann leider doch nicht für **jedes** Problem geeignet, sind aber doch immerhin auf jeden Fall einen Versuch wert. Doch wie genau funktioniert jetzt so ein genetischer Algorithmus?

Der Suchraum für unseren Algorithmus wird folgendermassen erzeugt: Wir stellen sämtliche Instanzen unserer Trainingsdaten als Vektoren dar. Diese Darstellung hilft uns dabei, daß der Suchraum letztlich ähnlich unserem ursprünglichen Suchraum ist. Die Trainingsdaten werden dann in positive und negative Beispiele eingeteilt.
Für genetische Algorithmen ist es wichtig, daß die Anzahl der Individuen, die der Algorithmus betrachten soll, gleichbleibend ist, ebenso wie die Anzahl der Merkmale der einzelnen Individuen. Als Ausgangs-Zahl der Individuen verwenden wir die Anzahl an Instanzen aus der Trainingsmenge. Die Anzahl der Merkmale jedes Individuums, ist gleichbleibend, da wir auch hier wieder die

bitweise Darstellung der Merkmale wählen, also "0", wenn das Merkmal in der jeweiligen Auswahl nicht vorhanden ist, und "1", wenn es doch vorhanden ist. Jedes Individuum wird nun anhand seiner Gesamt-Fitness durch eine Fitness-Funktion bewertet. Die Gesamt-Fitness ist hierbei wieder die Relevanz der Merkmals-Auswahl für unser eigentliches Klassifizierungsproblem.
Bisher unterscheidet sich der gesamte Vorgang noch nicht von den bereits vorgestellten Ansätzen. Der Teil, der die Bezeichnung "genetisch" rechtfertigt, ist folgender:
Nachdem die Fitness-Funktion jedem Individuum eine Gesamt-Fitness zugeordnet hat, werden die vielversprechendsten, also diejenigen Individuen mit einer hohen Fitness, miteinander gekreuzt. Dazu wird an einer beliebigen, jedoch bei beiden Individuen gleichen, Stelle ein Schnitt in der Bitmuster - Folge beider Individuen gemacht und die Bitmuster dann über Kreuz miteinander verbunden. Das Bild auf Seite 214 zeigt, wie der Vorgang vonstatten geht. Als zusätzliche Möglichkeit bietet sich die Mutation an, die bei einem auf eine solche Art neu entstandenen Kind ein beliebiges Bit zufällig umkippt, in der Hoffnung, daß durch diese Art Mutationen noch bessere Kinder erzeugt werden können, die durch einfache Kreuzung evtl. nie entstanden wären. Wichtig ist, da es ja gerade um die Verknüpfung von Selektion und Generierung geht, daß die Merkmale der Individuen auch aufgespalten werden können, falls es sich um generierte Merkmale handelt, solange die Gesamt-Anzahl an Merkmalen der Kinder wieder die gleiche ist, wie bei allen anderen Individuen. In Abbildung 12.5 ist zu sehen, wie ein System unter Verwendung eines genetischen Algorithmus aussehen kann. Dies ist allerdings natürlich wieder nur ein Ansatz, wie man die beiden Systeme miteinander verknüpfen kann. Die Effizienz der Vorgehensweise sollte allerdings intuitiv klarwerden.

12.5 Bezüge zu anderen Themen

Die Merkmalsselektion und -generierung ist ein wichtiger Bestandteil des gesamten KDD-Prozesses. Ohne die Selektion und Generierung von Merkmalen in der Vorverarbeitung ist es schwierig, zufriedenstellende Ergebnisse bei der Wissensentdeckung zu erzielen (vergl. 2). Dabei ist der ganze Prozess der Selektion und Generierung unabhängig zu sehen von dem verwendeten Lernverfahren. So kann die Selektion sowohl die Ergebnisse z.B. von Entscheidungsbäumen (vergl. 4) oder Support Vector Machines (vergl. 5).
Um das Ergebnis der Selektion und Generierung noch zu verbessern, kann die Auswahl anhand einer Kreuzvalidierung getestet werden, wie sie in Kapitel 3 vorgestellt werden.
Man wird bei einem Blick auf die KDD-Cups feststellen, daß es in der Tat we-

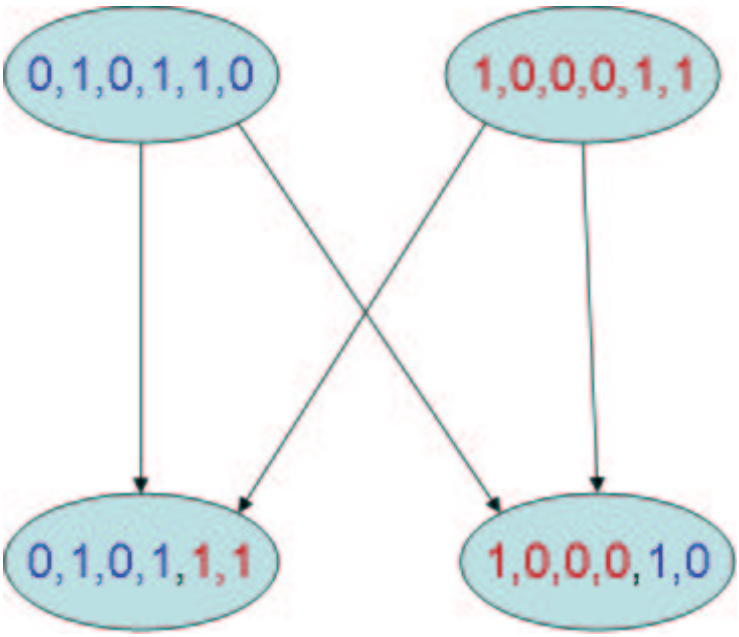


Abbildung 12.6: Beispielhafte Darstellung einer Kreuzung zweier Individuen.

nige Anwendungen von Data Mining gibt, in denen keine Merkmalsselektion betrieben wurde (vergl. refchap:ddcupwinners).

12.6 Fazit

Wir haben gesehen, daß eine Lernaufgabe wesentlich effizienter zu erledigen ist, wenn nicht alle Merkmale durch den Algorithmus betrachtet werden. außerdem ist deutlich geworden, daß unter Umständen einige Merkmale miteinander verknüpft werden müssen, damit eine Beziehung zwischen dem zu klassifizierenden Merkmal und den vorhandenen Merkmalen hergestellt werden kann. Schliesslich konnten wir noch sehen, wie die beiden Mechanismen miteinander verknüpft werden konnten. Abschliessend sei gesagt, daß der Computer alleine meist nicht in der Lage ist, eine ausreichend gute Auswahl an Merkmalen zu treffen, bzw. Merkmale zu generieren, die zu einer optimalen Klassifizierung führen, d.h. der Nutzer wird selber noch einen grossen Teil der Selektion und Generierung bzw. der Anpassung dabei verwendeten Parameter vornehmen müssen, und sei es nur, um die jeweils erfolgversprechendste Methode zu wählen.

Kapitel 13

Die Welt des KDD Cup - Ausgewählte Aufgaben und Ergebnisse alter KDD Cups

Marcel Gaspar

13.1 Einleitung

Der KDD Cup ist ein von der "Special Interest Group on Knowledge Discovery and Data Mining" (SIGKDD) der Firma ACM veranstalteter Wettbewerb im Bereich Wissensentdeckung. Er wird seit 1997 in Kombination mit der jährlich stattfindenden KDD-Konferenz der SIG auf internationalem Niveau und mit steigendem Interesse aus Forschung und Industrie ausgetragen.

Im vorliegenden Vortrag sollen die Aufgabenstellungen dieses Wettbewerbes der vergangenen Jahre und einige Lösungsvorschläge der jeweiligen Gewinner(inen) exemplarisch dargestellt werden. Es handelt sich hierbei um eine Zusammenfassung der Wettbewerbsberichte der von der SIGKDD veröffentlichten Zeitschrift "SIGKDD Explorations"[1], welche, neben den Homepages der entsprechenden Cups, ausschliesslich als Quelle dieses Vortrages verwendet wurde. Aus jedem Jahr seit 2000 werden alle Aufgabenstellung des jeweiligen Jahres

[1] (online verfügbar unter http://www.acm.org/sigkdd/explorations/)

vorgestellt, zusammen mit den Gewinnerlösungen von zwei der insgesamt drei Aufgaben des Cup 2001.

In den folgenden Abschnitten wird von der Tatsache ausgegangen, dass bei Veröffentlichung jeder Aufgabenstellung der dazugehörige Trainingsdatensatz den Teilnehmern und Teilnehmerinnen zum Download zur Verfügung gestellt wird. Nach einer Bearbeitungszeit von einigen Wochen wird der Testdatensatz veröffentlicht, auf welchen sich die erarbeiteten Lösungsstrategien beziehen sollen (Aussnahme: deklarative Aufgabestellungen, da kein Testdatensatz erforderlich). Die eingereichten Lösungen werden bezüglich einer auf jede Aufgabenstellung angepassten Kosten-/Bewertungsfunktion nach Rangliste eingestuft, und der Teilnehmer bzw. die Teilnehmerin mit der höchsten Bewertung gewinnt. Auf die Bewertungskriterien im Einzelnen wird in den Kapiteln noch genauer eingegangen. Die in den einzelnen Abschnitten angesprochenen Datensätze liegen alle in Form von Textdateien vor, mit einer Aneinanderreihung der jeweiligen (meistens kommaseparierten) Attributwerte.

13.2 Cup 2000

Der Cup 2000, veranstaltet in Boston, bestand aus insgesamt fünf Aufgaben zum Thema e-Commerce, die sich in folgende 2 Kategorien einteilen lassen.[2] Zur Charakterisierung der Lernaufgaben siehe Kapitel 2.

- überwachtes Lernen (Klassifikation mit 2 Klassen in Aufgabe 1 und mit 4 Klassen in Aufgabe 2)
- unüberwachtes Lernen in den Aufgaben 3, 4 und 5

Die einzelnen Aufgaben beruhen auf einem einzigen Datensatz, auf den im Folgenden zuerst eingegangen wird, danach werden die Aufgaben mit den jeweiligen Bewertungsstrategien vorgestellt.

13.2.1 Der Datensatz

Als Datensatz des Cup 2000 dient eine über 2 Monate protokollierte Datenbank eines amerikanischen Online Versandhandels (Gazelle.com) mit Seitenklick-, Bestell- und Kundeninformationen. Der Datensatz liegt in zwei Versionen vor, nämlich multirelational (2 Tabellen mit Querverweisen als Attribute) und aggregiert (insgesamt drei flache Tabellen ohne Querverweise).

Die erste Tabelle ("Clickstream"-Tabelle) der nicht aggregierten Version des Datensatzes besteht aus den Attributen "Customer Profiles", "Customer Infos", "Web Sessions", "Products", "Assortments" und "Contents" in Form von Querverweisen, zuzüglich drei Zielvariablen. Ein Eintrag dieser Tabelle stellt dabei jeweils eine aufgerufene Seite dar, was bedeutet, dass sich bei Sitzungen, die mehrere Seiten umfassen, einige der oben erwähnten Attribute wiederholen. Die zweite Tabelle ("Orderlines"-Tabelle) enthält die Attribute "Customer Profiles", "Customer Infos", "Order headers", "Products", "Assortments" und "Promotions" wieder in Form von Querverweisen. Ein Eintrag dieser Tabelle stellt dabei jeweils einen auf Gazelle.com bestellten Artikel dar, was bedeutet, dass sich bei Bestellungen, die mehrere Artikel umfassen, einige der oben erwähnten Attribute wiederholen. Auf die inhaltliche Bedeutung der einzelnen Attribute wird hier nicht näher eingegangen, da diese für die Bearbeitung der Aufgaben irrelevant ist.

[2] vergl [48]

Die aggregierte Version des Datensatzes besteht aus insgesamt 3 Tabellen, jeweils eine für die Aufgaben 1 und 4 zusammen, eine für die Aufgaben 2 und 5 zusammen, und eine für Aufgabe 3. Vereinfacht ausgedrückt stellt ein Eintrag in den Tabellen der Aufgaben 1/4 und Aufgaben 2/5 nun die Zusammenfassung der zu einer Session gehörigen betrachteten Seiten der Clickstream-Tabelle dar, was bedeutet, dass bedeutend weniger Zeilen vorhanden sind, aber zusätzliche Attribute zur Berücksichtigung der eigentlich multirelationalen Struktur der Clickstream-Tabelle eingeführt werden. Auf ähnliche Weise stellt ein Eintrag der Tabelle zu Aufgabe 3 nun die Zusammenfassung der zu einer Bestellung gehörigen Einzelartikel der Orderlines-Tabelle dar. Tabelle 1 gibt einen Überblick über die Anzahl der Attribute und Beispiele der jeweiligen Datensatzversionen für die einzelnen Aufgaben.

	Trainingsdatensatz		Testdatensatz	
Aufgabe	Attribute	Zeilen	Attribute	Zeilen
1,4: unaggregated	217	777480	215	164364
2,5: unaggregated	217	777480	215	142204
3: unaggregated	232	3465	entfällt	entfällt
1,4: aggregated	296	234954	296	50558
2,5: aggregated	299	234954	296	62913
3: aggregated	518	1781	entfällt	entfällt

Tabelle 13.1: Überblick Datensatz für Aufgaben 1-5

Weiterhin liegt zur Bearbeitung der Aufgaben der Marketingkalender des Versandhandels vor, um auf möglicherweise stattgefundene Werbemassnahmen oder Layoutveränderungen der Homepage oder ähnliches, und damit verbundene mögliche Fluktuationen der Attributwerte des Datensatzes, hinzuweisen.

13.2.2 Die Aufgaben

- Aufgabe 1: Bei gegebener Menge betrachteter Seiten soll entschieden werden, ob der Besucher eine weitere Seite betrachtet oder die Sitzung sofort beendet

- Aufgabe 2: Bei gegebener Menge betrachteter Seiten soll entschieden werden, welche Produktkategorie von insgesamt vier (*Hanes, Donna Karen, American Essentials, Other*) sich der Besucher während des restlichen Teils der Sitzung noch anschauen wird

Die Bewertung der Aufgabe 1 erfolgt durch Ermittlung der Anzahl der auf einem Testdatensatz korrekten Vorhersagen.

Für Aufgabe 2 erfolgt die Bewertung durch eine gewichtete Aufsummierung korrekter Vorhersagen nach folgenden Kriterien: zwei Punkte für jede Vorhersage einer speziellen Kategorie (alles ausser *Other*), welche in der restlichen Sitzung tatsächlich noch besucht wird; einen Punkt für jede Vorhersage von *Other*, nach welcher in der restlichen Sitzung tatsächlich keine der anderen drei speziellen Kategorien mehr besucht wird; null Punkte für jede andere Vorhersage.

- Aufgabe 3: Charakterisierung der Kunden, die bei einer durchschnittlichen Bestellung mehr als 12 Dollar ausgaben
- Aufgabe 4: Charakterisierung der Seiten, bei denen Kunden/Besucher die Sitzung abbrechen
- Aufgabe 5: Charakterisierung der Produktkategorien (*Hanes, Donna Karen, American Essentials, Other*), die Kunden/Besucher bei gegebener Menge betrachteter Seiten während der restlichen Sitzung noch besuchen werden

Die Lösung sollte in Text- und Diagrammform abgegeben werden, in der Art, dass ein Geschäftsmann die ermittelten Resultate als verständlich und nützlich empfindet.

Die Bewertung der Lösungen dieser Aufgaben erfolgt erstens in Abhängigkeit von der Qualität (1-10) der Präsentation der ermittelten Resultate (P), zweitens in Abhängigkeit von der Korrektheit (1-10) (C), und drittens in Abhängigkeit von der Anzahl der für jede Frage ermittelten Resultate (Zusammenhänge), wobei die Punkte für jedes dieser einzelnen Resultate abhängt von der Qualität der Entdeckung (I: 0=nicht entdeckt, 1=teilweise entdeckt, 2=vollständig entdeckt) multipliziert mit der Gewichtung des Resultats (w).

Es ergibt sich folgende Bewertungsformel:

Gesamtpunkte = 3P + 3C + $\sum_{i=1}^{N} w_i I_i$

13.3 Cup 2001

Der Cup 2001, veranstaltet in San Francisco und im Zeichen der Bioinformatik stehend, bestand aus den folgenden drei Aufgaben des überwachten Lernens:[3]

- Teil 1: Klassifikation der Aktivität von Molekülen in der Medikamentenentwicklung
- Teil 2: Klassifikation der Funktion von Genen/Proteinen
- Teil 3: Klassifikation des Wirkortes von Genen/Proteinen

Die Aufgaben beruhen auf zwei verschiedenen Datensätzen, einem für Aufgabe 1 und einem für die Aufgaben 2 und 3. Auf den jeweiligen Datensatz wird in den Abschnitten zu den Aufgaben eingegangen. Weiterhin werden für Aufgabe 1 und 3 die Lösungen der Gewinner exemplarisch dargestellt.

13.3.1 Teil 1 - Klassifikation der Aktivität von Molekülen in der Medikamentenentwicklung

Ein wichtiger -und sehr arbeitsintensiver- Schritt in der Entwicklung neuer Medikamente stellt die Überprüfung und Analyse der Bindungsfähigkeit der jeweiligen synthetisierten Moleküle an einen gewünschten Rezeptor dar. In diesem Sinne enthält die Trainingsdatenbank für die erste Aufgabe des Cup 01 1909 verschiedene synthetisierte Moleküle, für welche pro Molekül in einem Merkmalsvektor von 139351 binärwertigen Attributen die dreidimensionalen Eigenschaften gespeichert sind, und dazu eine Kategorisierung des jeweiligen Moleküls bezüglich seiner Bindungseigenschaft (Aktivität A) an das Enzym Thrombin (Thrombin ist ein wichtiger Bestandteil des menschlichen Blutgerinnungssystems).

Dazu anzumerken bleibt, dass sich nur 42 der insgesamt 1909 Moleküle an Thrombin binden, und die Aufgabe stellt somit einen Versuch der Ermittlung jener Merkmale dar, die tatsächlich die Bindungseigenschaft an einen gewünschten Rezeptor beeinflussen.

[3] vergl [14]

Davon ausgehend soll nun für 634 neuartige und unkategorisierte Moleküle in einer Testdatenbank vorrausgesagt werden, ob sich diese an Thrombin binden oder nicht (a = aktiv/bindend, i = inaktiv/nicht bindend)

Die Bewertung der Aufgabe erfolgt mittels ungewichteten Durchschnitts der Genauigkeit wahrer aktiv- und inaktiv- Vorhersagen auf den Verbindungen der Testdatenbank. Die Einbeziehung wahrer inaktiv-Vorhersagen in das Bewertungsmass begründet sich in der Tatsache, dass bei einem derart geringen Anteil aktiver Verbindungen ansonsten eine Klassifikation sämtlicher Moleküle als inaktiv zwar eine sehr hohe Genauigkeit erreicht, aber eine völlig unbrauchbare Lösung liefert.

13.3.2 Teil 1 - Lösung

Das beste Ergebnis dieser Aufgabe erreichte Jie Cheng mit der im Folgenden zusammengefassten Strategie:

1. Ausschluss von separaten Validierungsmengen oder Kreuzvalidierung aufgrund der geringen Anzahl aktiver Moleküle (42 von 1909).

2. Reduzierung der Dimensionen aufgrund der zu grossen Merkmalsmenge von 139351.

   Ermittlung des *Information Gain* I(m,A) für alle Merkmale m und die abhängige Variable A. Berücksichtigung aller Merkmale m, für die I(m,A) mindestens 0,035 beträgt. 200 Merkmale bleiben übrig.

   $I(A,B) = \sum_{a,b} P(a,b) \log \frac{P(a,b)}{P(a)P(b)}$

3. Erzeugen von 5 Bayes'schen Netzen unterschiedlicher Komplexität mit den 200 in Punkt 2. identifizierten Merkmalen. Verwendet wurde dafür das selbst erstellte und frei verfügbare Werkzeug "BN Powerpredictor".

4. Analyse der Markov Decke des Zielknotens (Knoten der abhängigen Variable A) und Ignorieren aller Knoten ausserhalb der Decke, dadurch Reduzierung der Anzahl der benötigten Merkmale der 5 Netze auf 2 bis 12.
   Die Markov Decke eines Knotens K beinhaltet alle Elternknoten von K, alle Nachfolgeknoten von K und alle Elternknoten der Nachfolgeknoten von K. Die Bedeutung dieser Knotenmenge liegt in der Tatsache, dass die Knoten der Markov Decke eines Knotens K in einem Bayes'schen Netz eindeutig den Wert von K bestimmen.

5. Messung der Flächen unter den ROC-Kurven der 5 Netze bei Anwendung auf Trainingsdatensatz. Für eine Definition von ROC-Kurven siehe Kapitel 3. Auswahl des Netzes mit bestem Verhältnis zwischen Ergebnis und Anzahl benötigter Knoten (gegen die Auswahl des tatsächlich besten Netzes spricht die Möglichkeit der Überanpassung, weshalb die Einfachheit des Modells erhöhte Berücksichtigung fand, auf das genaue Verhältnis wird in den Quellen jedoch nicht eingegangen). 1 Netz mit 4 Merkmalen bleibt übrig.

6. Anwendung des Netzes auf die 634 Moleküle des Testdatensatzes.

   Problem: Bestimmung des Cut-Points zwischen aktiv- und inaktiv- Vorhersagen zur Klassifikation einer Eingabe mittels Wahrscheinlichkeitswert am Zielknoten.

   Lösung: Es existieren nur 9 verschiedene Werte am Zielknoten für alle 634 Eingaben. Es gibt somit nur 8 mögliche cut-points, mit welchen das Netz jeweils 32, 71, 72, 74, 75, 215, 223 oder 550 Moleküle als aktiv klassifizieren würde.

   Wahl jenes cut-points, mit dem 223 Moleküle als aktiv klassifiziert werden, und Klassifikation der Knoten mittels diesem. Die Auswahl des cut-points wurde dabei -neben dem Raten- beeinflusst durch folgende Überlegungen: trotz anderer Verteilung im Testdatensatz existieren immernoch mehr inaktive als aktive Fälle, und Falschklassifizierung eines tatsächlich aktiven Falles erzeugt mehr Kosten.

13.3.3 Teil 2 - Klassifikation der Funktion von Genen/Proteinen

Ein Gen stellt vereinfacht ausgedrückt den Bauplan für ein bestimmtes Protein dar. Einer der Forschungsschwerpunkte der Genetik ist die Analyse der Gene eines Organismus bezüglich des durch ein bestimmtes Gen codierten Proteins, dessen Funktion und des Ortes in einer Zelle, an dem das Protein wirkt. Die zweite Aufgabe des Cup 01 verfolgt dieses Ziel auf folgende Weise: der Trainingsdatensatz besteht aus 862 bezüglich Funktion und Wirkort des jeweiligen codierten Proteins analysierten Genen des Hefe. Der Datensatz liegt in zwei Versionen vor: erstens in Form von 2 Tabellen ("Genes_relation" und "Interactions_relation") und zweitens als eine einzige (aggregierte) flache Tabelle ("Full_File"). Ein Eintrag der Tabelle Genes_relation repräsentiert die Eigenschaften eines Genes durch die folgenden 6 Attribute "Chromosomnummer", "Lebensnotwendigkeit", "Phenotypus", "Proteinklasse" Proteincharakteristika", "Proteinverbindung" zuzüglich des abhänigen Attributs "Function". Die Funktion ist dabei eine Teilmenge aus 15 möglichen Funktionen. Ein Eintrag der

Tabelle Interactions_relation enthält ein Paar wechselwirkender Gene, die Art der Wechselwirkung (physisch, genetisch, physisch+genetisch) und die Stärke der Wechselwirkung als reelle Zahl zwischen 0 und 1. Gene, die mit mehreren anderen Genen wechselwirken, treten dabei aufgrund der paarweisen Darstellung wiederholt auf. Die Tabelle Full_File stellt eine Art Zusammenfassung der beiden ersten Tabellen für eine Verwendung mit nicht-relationalen Lernverfahren dar. Sie enthält für jedes Gen eine Zeile, und 3000 Spalten mit Merkmalen, die aus der Eigenschafts- und der Wechselwirkungs-Tabelle gewonnen wurden.

Die Aufgabe besteht nun darin, für 381 gegebene neue Gene der Testdatenbank vorherzusagen, welche Funktion(en) jedes Gen besitzt (bzw. das durch dieses Gen repräsentierte Protein).

Die Bewertung der Aufgabe ergibt sich als die Anzahl der korrekt vorhergesagten (Gen, Funktion)-Relationen unter Abzug von falsch vorhergesagten bzw. fehlenden Relationen. Der Abzug von Falschaussagen ist nötig, da ein Gen mehrere Funktionen besitzen kann, wovon möglichst viele berücksichtigt werden sollen.

13.3.4 Teil 3 - Klassifikation des Wirkortes von Genen/Proteinen

Die Motivation und der Datensatz für Aufgabe 3 sind identisch zur Aufgabe 2 mit dem einzigen Unterschied, dass das abhängige Attribut nun nicht die Funktion ist, sondern der Wirkort "Localization" des codierten Proteins. Der Ort ist dabei genau einer aus 15 möglichen Orten innerhalb einer Zelle.

Die Aufgabe besteht darin, für 381 gegebene neue Gene der Testdatenbank aus Aufgabe 2 voherzusagen, an welchem Ort in einer Zelle das durch das jeweilige Gen codierte Protein wirkt.

Die Bewertung ergibt sich als die Anzahl der korrekt vorhergesagten (Gen, Wirkort)-Relationen (genau eine pro Gen).

13.3.5 Teil 3 - Lösung

Die beste Lösung dieser Aufgabe lieferten Hayashi, Sese und Morishita mit der im Folgenden zusammengefassten Nearest-Neighbor-Strategie:

1. Vorüberlegungen:

   Sehr viele Attributwerte fehlen.

   Wechselwirkende Gene (bzw. die erzeugten Proteine) befinden sich gewöhnlich am selben Ort.

   Signifikante Korrelation der Attribute Class (Proteinklasse), Complex (Proteinverbindung), Motif (Proteincharakteristika) und Location (Wirkort); keine signifikante Korrelation zwischen den restlichen Attributen.

   Von den Merkmalen Class, Complex, Motif und Wechselwirkung ist - trotz vieler fehlender Werte- immer mindestens eines gegeben für alle Gene der Trainingsdatenbank und 367 Gene der Testdatenbank.

   Aufgrund dieser vier Feststellungen im Folgenden besondere Berücksichtigung der 4 Merkmale Class, Complex, Motif und Wechselwirkung.

2. Anwendung der Nächste-Nachbar-Strategie.

   Problem: Definition der Einträge/Gene, die zu einer Eingabeinstanz "benachbart" -bzw dieser "ähnlich"- sind.

   Lösung: zwei Instanzen sind genau dann benachbart, wenn ihre Werte für bestimmte gegebene Attribute übereinstimmen, welche in einer Liste L nach Priorität angeordnet sind.

3. Bei gegebener Liste $[a_1...a_n]$ von solchen nach Priorität geordneten Attributen, die bei benachbarten Instanzen übereinstimmen sollen, lässt sich für ein Gen r des Testdatensatzes die Menge NN der nächsten Nachbarn von r bestimmen. Der Wirkort von r ergibt sich dann als Mehrheitsentscheid der Wirkorte aller nächsten Nachbarn, und die Genauigkeit des Verfahrens errechnet sich als Quotient aus der Anzahl der auf diese Weise (in Abhängigkeit von der benutzten Attributliste) korrekt bestimmter Gene r durch die Anzahl aller Gene des Testdatensatzes:

   $$accuracy\left([a_1...a_n]\right) = \frac{|\{r \in TestSet || Label(r) = predict(r, NN, [a_1...a_n])\}|}{|\{r \in TestSet\}|}$$

   Das Problem stellt nun also die Erzeugung einer solchen Attributliste dar, unter Maximierung der Genauigkeit des Mehrheitsentscheides mittels der Menge nächster Nachbarn, welche ja zuvor durch die Attributliste erzeugt wurde. In [14] befindet sich ein Beweis der NP-Vollständigkeit dieses Optimierungsproblems, auf den hier nicht näher eingegangen wird. Als Lösung führen Hayashi, Sese und Morishita eine branch-and-bound Suche in der Menge aller möglichen Attributlisten durch.

   Für eine Betrachtung der Algorithmen zur Berechnung der Menge NN von nächsten Nachbarn zur Eingabeinstanz r, und zur Berechnung einer optimalen Attributliste sei an dieser Stelle auf [14] verwiesen, oder auf die zum Vortrag gehörigen Folien Nr. 30 und 31.

13.4 Cup 2002

Der Cup 2002, veranstaltet in Edmonton und erneut von der Bioinformatik geprägt, bestand aus zwei Aufgaben:[4]

- Teil 1: Informationsextraktion aus biomedizinischen Artikeln
- Teil 2: Klassifikation des Einflusses von Genen

Jede der beiden Aufgaben besitzt ihren eigenen Datensatz, auf welchen wieder in den Abschnitten eingegangen wird.

13.4.1 Teil 1 - Informationsextraktion aus biomedizinischen Artikeln

Molekularbiologisches Wissen existiert -genau wie viele andere wissenschaftliche Erkenntnisse auch- auf der einen Seite als weitgehend unstrukturierte Forschungsliteratur in Form von unzähligen Texten, auf der andere Seite strukturiert bzw teilstrukturiert in Form von Datenbanken. Als zeitraubendes und arbeitsintensives Problem erweist sich nun die Auswahl einer Teilmenge von Dokumenten aus einer Menge gegebener Dokumente, die für eine Aktualisierung der Datenbanken mit Inhalten der Forschungsliteratur geeignet sind. Diesem Problem geht die erste Aufgabe des Cup 02 nach. Als Menge wissenschaftlicher Texte dienen 1075 Dokumente im Tex-Format über die Genetik der Fruchtfliege, zuzüglich einer Liste erwähnter Gene für jedes der Dokumente. Auf der anderen Seite dient www.FlyBase.org als Datenbank, und beinhaltet aktuelle Forschungsergebnisse über das Genom der Fruchtfliege. Eine Teilmenge von 862 Dokumenten dient als Traingsdatensatz, die restlichen 213 als Testdatensatz.

Das zentrale FlyBase Kriterium ist nun die Frage, ob ein gegebenes Dokument experimentelle Ergebnisse über die Gen-Erzeugnisse der Fruchtfliege enthält, und identifiziert solche Dokumente, deren Inhalte und Ergebnisse für eine Aktualisierung der Datenbank in Betracht gezogen werden können. Die resultierende Aufgabe besteht aus drei Teilen:

[4] vergl [71]

1. Ausgabe einer Rangliste der Dokumente bezüglich der Erfüllung des "FlyBase Kriteriums"

2. J/N Entscheidung für jedes Dokument, ob es das Kriterium derart erfüllt, dass es für die Aktualisierung der DB geeignet ist

3. J/N Entscheidung für jedes Gen aus den Listen der erwähnten Gene jedes Dokumentes, ob es für die Aktualisierung der DB geeignet ist

Die Bewertung der Aufgabe ergibt sich als Summe der Punkte für die drei Teilaufgaben: ROC-Kurve für Teil 1 + F-Test für Teil 2 + F-Test für Teil 3 für die Vorhersage auf den 213 Texten des Testdatensatzes. Für eine Definition von ROC-Kurve und F-Test siehe Kapitel 3.

13.4.2 Teil 2 - Klassifikation des Einflusses von Genen

Komplexe Wirkmechanismen in einer Zelle werden oft bestimmt bzw. beeinflusst von einer Vielzahl von Genen des Organismus. Der Frage, welchen Einfluss ein bestimmtes Gen auf ein verborgenes (unbestimmtes) Wirksystem in einer Zelle hat, versuchte der Cup 02 mit der zweiten Aufgabenstellung nachzugehen. Ohne Anspruch auf biologische Vollständigkeit wird an dieser Stelle nur die Idee im Überblick dargestellt.

Der verwendete Datensatz, die sogen. "Yeast Deletion Library", enthält 20000 verschiedene Arten der Hefe, welche sich untereinander darin unterscheiden, dass in jeder Art ein bestimmtes Gen deaktiviert ist. Um nun den Einfluss der Gene auf ein verborgenes Wirksystem bestimmen zu können, muss ein derartiges (messbares) System erstmal identifiziert werden. Als solches System wurde der sogen. "AHR Signaling Pathway" gewählt, und da das AHR Protein ursprünglich nicht zur Hefe gehört, wurden 4507 Hefearten der Yeast Deletion Library um das entsprechende Gen erweitert zuzüglich eines Reporter-Systems zur Messung der Aktivität des AHR innerhalb einer Zelle der jeweiligen Hefeart. (AHR = Aryl Hydrocarbon Receptor = Protein, das u.a. die Reaktion einer Zelle auf Schadstoffe vermittelt). Anzumerken bleibt, dass nur 127 Arten (also Gene) einen Einfluss auf die Aktivität zeigten, die restlichen 4380 Gene zeigten keinen Einfluss. Von der um AHR erweiterten Yeast Deletion Library werden 3000 Einträge als Trainingsdatensatz und der Rest als Testdatensatz verwendet. Um Domänenwissen der Teilnehmer(innen) auszuschliessen, blieb die gerade beschriebene Art des betrachteten Wirksystems bis zum Ende des Cups unbekannt.

Der Datensatz besteht aus folgenden drei Teilen:

1. Hierarchische Repräsentation der Zusammenhänge zwischen Genen und 1. Proteinklassen (pc.txt, pc-hierarchy.txt), 2. Funktionen (function.txt, function-hierarchy.txt) und 3. Wirkorten (localization.txt, localization-hierarchy.txt)

2. Tabelle mit Paaren von sich physikalisch gegenseitig beeinflussenden Proteinen

3. 15235 Abstracts von molekularbiologischen Texten mit Index, in welchem Abstract ein bestimmtes Gen erwähnt ist

Die Aufgabe besteht nun darin, für ein gegebenes Gen des Testdatensatzes den Einfluss auf die Aktivität des Wirksystem AHR vorherzusagen: *change* = Einfluss nur auf AHR (AHR-spezifisches Gen); *no change* = kein Einfluss auf irgendwas; *control* = Einfluss auf AHR und auf ein Kontrollwirksystem (nicht AHR-spezifisches Gen)

Die Bewertung der Aufgabe erfolgt durch Addition der Flächen unter den Roc-Kurven der jeweiligen Ergebnisse für die beiden Vorhersagefälle, in denen die Kontrollbedingung einmal berücksichtigt wird und einmal nicht.

13.5 Cup 2003

Der Cup 2003, veranstaltet in Washington, verfolgte das Thema des Network-Mining und stellte insgesamt 4 Aufgaben:[5]

- Teil 1: "citation prediction"
- Teil 2: "data cleaning"
- Teil 3: "download estimation"
- Teil 4: "open task"

Da der Datensatz für alle 4 Aufgaben ähnlich ist, wird auf diesen im Vornherein eingegangen, und anschliessend die Aufgabenstellung kurz erläutert.

13.5.1 Der Datensatz

Als Datensatzquelle für alle 4 Aufgaben dient arXiv.org, ein Online-Archiv mit derzeit rund 233000 wissenschaftlichen Texten vorrangig aus den Bereichen Physik und Mathematik. Weiterhin charakteristisch für arXiv.org sind der Zuwachs von 40000 neuen Dokumenten pro Jahr, 10 Millionen Anfragen pro Monat, durschnittlich rund 300 Downloads pro Dokument, Protokollierung seit 1993, und ein interner Zitierungsgraph (Knoten = Dokumente, Kanten = Verweise bzw Zitierungen zwischen den Dokumenten).

Als Datensatz für die Aufgaben 1,3 und 4 dient die Teilmenge HEP-TH aller Texte über Hochenergiephysik-Theorie (30000 Dokumente von 57448 Autoren, 355000 interne Zitierungen). Die Texte sind charakterisiert durch eine eindeutige arXiv-ID und liegen im TeX-Quellcode -vorhandene Graphiken entferntvor, zuzüglich des auf HEP-TH eingeschränkten Zitierungsgraphen. Weiterhin existiert für jedes Dokument das tatsächliche Erstellungsdatum (da das tatsächliche Erstellungsdatum vom arXiv-Veröffentlichungsdatum abweichen kann), und eine Zusammenfassung des Textes mit den Informationen "Erstellungsdatum", "Korrekturdatum", "Titel", "Autor(en)" und "Inhalt".

Für Aufgabe 3 wird neben den oben beschriebenen Daten zusätzlich das Download-Log der ersten 60 Tage nach Veröffentlichung benötigt für alle Dokumente, die

[5] vergl [1]

im Februar/März 2000 oder Februar/April 2001 oder März/April 2002 in arXiv veröffentlicht wurden.

Den Datensatz für Aufgabe 2 stellt die Teilmenge HEP-PH aller Texte über Hochenergie-Experimentalphysik dar (35000 Dokumente), wieder im TeX-Quellcode vorliegend.

13.5.2 Die Aufgaben

1. Aufgabe 1: Vorhersage von (d, Z) für jedes Dokument d aus dem Datensatz zu Aufgabe 1,

   mit Z = ((Anzahl Zitierungen des Dokumentes von Mai bis Juli) - (Anzahl Zitierungen des Dokumentes von Februar bis April)).

   Es soll also der Zuwachs an Zitierungen vorhergesagt werden. Die Bewertung erfolgt über die Anzahl der korrekt vorhergesagten (d, Z) Paare.

2. Aufgabe 2: Erstellung eines Zitierungsgraphen G = (V, Z) auf den Dokumenten aus HEP-PH, ausgegeben als zweispaltige Textdatei (sortierte Adjazenzliste). Die Bewertung erfolgt über die Anzahl korrekt vorhergesagter Kanten (Zitierungen zwischen zwei Dokumenten).

3. Aufgabe 3: Vorhersage von (d, D) für jedes Dokument d aus dem Datensatz zu Aufgabe 3, das im April 2000 oder März 2001 oder Februar 2002 veröffentlicht wurde,

   mit D = Anzahl Downloads in den ersten 60 Tagen. Die Bewertung erfolgt über die Anzahl der korrekt vorhergesagten (d, D) Paare.

4. Aufgabe 4: Definition einer eigenen Fragestellung und Präsentation der zugehörigen Lösung. Die Bewertung erfolgt durch die Inhaber des KDD-Cup co-chairs (Mark Craven, David Page, Soumen Chakrabarti) in Hinblick auf die Neuartigkeit der Ergebnisse, Korrektheit von verwendeten Methoden und Auswertungen, und der Wichtigkeit des Ergebnisses für arXiv.org.

13.6 Bezug zu den anderen Themen

Der Bezug dieses Vortrages zu anderen Vorträgen der Seminarphase ist generell nur dadurch gegeben, das die in den anderen Vorträgen vorgestellten Verfahren, Konzepte und Theorien bei der Lösung bestimmter Aufgabenstellungen des KDD-Cups eine Anwendung finden können oder nicht. Da in diesem Vortrag über die Welt des KDD-Cup natürlich nicht sämtliche in der Vergangenheit verwendete Lösungensstrategien dargeboten werden können, und die anderen Vorträge wiederum nicht alle möglichen zum KDD und DM gehörigen Konzepte abdecken, existiert nur eine relativ geringe gegenseitige Bezugnahme.

Der KDD-Prozess (Kapitel 2) Der Bezug zum Vortrag über den KDD-Prozess liegt darin, das für jede der hier dargestellten Aufgabenstellungen Ausschnitte des KDD-Prozesses nachmodelliert werden müssen. Einige Phasen des Wissenenteckungsprozesses entfallen dabei aber normalerweise aufgrund der Art der Aufgabenstellung und der Art zur Verfügung stehenden Datensätze, so z.B. das Unternehmensverständnis.

Handwerkszeug der Statistik (Kapitel 3) Der Bezug zu sämtlichen Teilen des Vortrages über statistisches Handwerkszeug ist ausnahmslos gegeben durch die Forderung nach Methoden zur Bewertung einer eingesetzten Lösungsstrategie, zur Analyse von Datensätzen, zum Vorverarbeiten von Datensätzen, zum Einschätzen der Qualität von Vorhersagen, usw.

Top down induction of decision trees (Kapitel 4) Das Konzept der Entscheidungsbäume findet in diesem Vortrag keine Anwendung, wird aber bei vielen anderen hier nicht vorgestellten Lösungen des KDD-Cup häufig als einer der ersten Schritte eingesetzt, so z.B. in der Gewinnerlösung der Aufgabe 1 des Cup 99 (vergl. [58]), um z.B. ein besseres Verständnis für die Daten zu gewinnen.

Den Bezug zu den Themen **Support Vector Machines** (Kapitel 5), **Regellernen mit RDT** (Kapitel 6), **Subgruppenentdeckung** (Kapitel 7), **Apriori** (Kapitel 8), **Apriori für zeitliche Daten** (Kapitel 9) und **Clustering** (Kapitel 10) zeigen beispielhaft die folgenden Diagramme, welche die zur Bearbeitung der Aufgaben des Cup 2000 und 2001 verwendeten Algorithmen darstellen. Die einzelnen Verfahren wurden in den in diesem Vortrag

vorgestellten Lösungen nicht erwähnt oder gebraucht, werden jedoch häufig in anderen Lösungstrategien bzw. bei anderen Lernaufgaben eingesetzt.

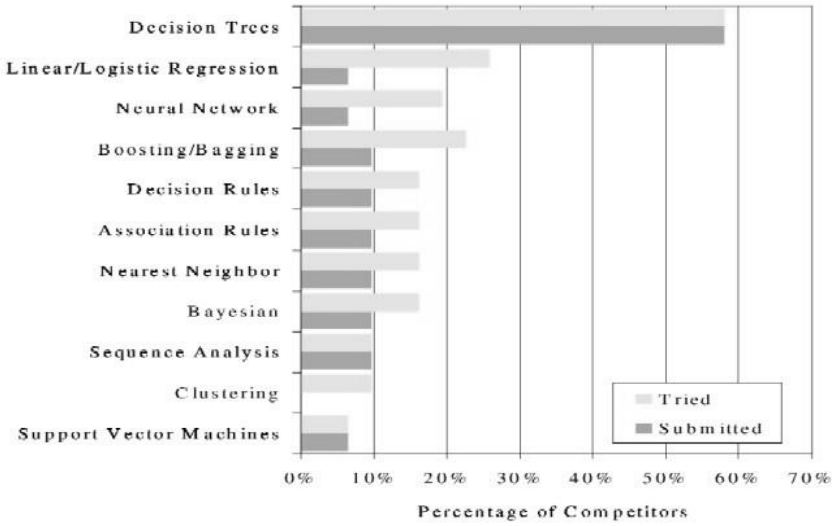


Abb.1: verwendete Verfahren Cup 00

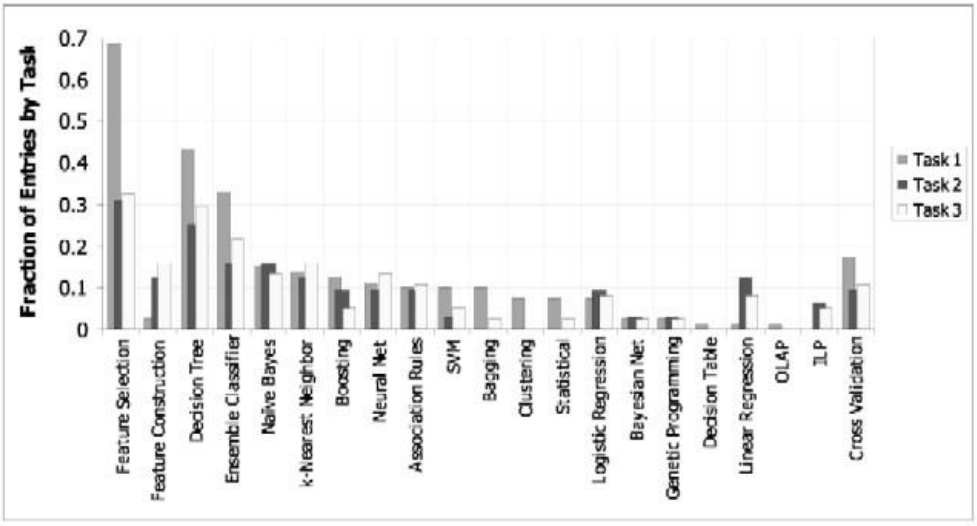


Abb.2: verwendete Verfahren Cup 01

Bagging and Boosting (Kapitel 11) Von Methoden des Bagging und Boosting wurde z.B. in der Gewinnerlösung der ersten Aufgabe des Cup 99 (Classifier Learning Contest) Gebrauch gemacht (vergl. [58]), worauf in dieser Ausarbeitung jedoch aus Platzgründen nicht eingegangen wurde.

Merkmalsgenerierung und -selektion (Kapitel 12) Das Konzept der Merkmalsgenerierung und -selektion wird in annähernd allen Lösungen, egal ob in diesem Vortrag beschrieben oder nicht, unterschiedlich exzessiv eingesetzt. So wurde z.B. in der Lösung zur Aufgabe 1 des Cup 01 mit Hilfe des Masses "Information Gain" eine Auswahl von 200 relevanten Merkmalen aus insgesamt 139351 durchgeführt. Weiterhin wählte die Lösung von Aufgabe 3 des Cup 01 durch geeignete Definition des Begriffs "Nachbarschaft" vier (nach Priorität geordnete) Merkmale von insgesamt sieben -auf zwei Tabellen verteilten- Merkmalen aus.

Merkmalsgenerierung wurde in den Cups 00/01 vor Veröffentlichung der Daten durchgeführt, indem die multirelationalen Datensätze nach einem in den entsprechenenden Quellen leider nicht genauer beschriebenden Verfahren zu einer einzigen Tabelle zusammengefasst wurden (aggregierte Version der Datensätze).

Teil II

Siegerlösungen vergangener KDD-Cups

Kapitel 14

KDD-Cup 1998

Holger Flick, Daniel Hakenjos, Felix Jungermann, Siehyun Strobel

14.1 Musterlösung und Datenexploration

14.1.1 Einführung

Der KDD-Cup 1998 beschäftigte sich mit der Aufgabe, für eine Spendenorganisation eine Vorhersage bzgl. der Spendenhöhe bei einer Spendenaktion von Juni 1997 zu treffen.

Nachmodelliert wurde eine Lösung von Jim Georges und Anne H. Milley (SAS Institute Inc.)[42], die mit ihrem Beitrag zwar nicht am KDD-Cup 1998 teilnahmen, jedoch mit ihrer Lösung ein besseres Ergebnis erreichte, als der Erstplazierte im Wettbewerb.

Es wurde versucht, die Lösung möglichst nah nachzumodellieren. An manchen Stellen war dies allerdings aufgrund fehlender Informationen zu in dem Artikel erwähnten Methoden und Werkzeugen nicht immer möglich, so dass versucht wurde, mit äquivalenten oder ähnlichen Methoden ähnliche Ergebnisse zu erzielen. Die konkreten Probleme bei der Nachmodellierung werden

an den entsprechenden Schritten ausführlich betrachtet.

14.1.2 Aufgabenstellung

Die Lernaufgabe des KDD Cups 1998 besteht darin vorherzusagen, welcher Spender bei der Spendenaktion im Juni 1997 tatsächlich spenden wird - es handelt sich also um eine Klassifikationsaufgabe. Durch diese Klassifikation kann der eingenommene Spendenbetrag berechnet werden. Die Testdatenmenge besteht aus 95412 Datensätzen mit 481 Attributen. Jeder Datensatz enthält demographische, soziale und persönliche Informationen zu einer Person, sowie die jeweilige Spendenhistorie.

Der Gewinner der Cups (Urban Science Applications, Inc.) erzielte einen Spendenbetrag von insgesamt $ 14712.-.

14.1.3 Modell der Musterlösung

Nach Exploration der Daten wurden Unregelmäßigkeiten in den Daten festgestellt. So konnten z.B. folgende Unstimmmigkeiten in den Jahrgängen gezeigt werden:

- Die Verteilung der geraden Jahrgängen unterscheiden sich erheblich zu den ungerade Jahrgängen.
- Es sind Spitzen in den Jahrgängen 1910, 1920, 1930, ..., 1990 erkennbar.
- Es lassen sich Jahrgänge bis einschließlich 1997 finden.

Diese Datenanomalien weisen auf generelle Probleme bzgl. der Datenintegrität hin.

Auch die genauere Betrachtung des Attributs *CONTROLN*, das in der Regel nur eine ID-Funktion besitzen sollte, weist darauf hin, dass die Kontrollnummer nicht sequentiell oder zufällig verteilt wurde, da Zusammenhänge zwischen diesem und anderen Attributen nachweisbar sind.

Jim Georges und Anne H. Milley setzten zur Vorhersage des Spendenbetrags ein sog. *Two-Stage Prediciton Model* ein. Es wurden die Spendenwahrschein-

lichkeit und die zu erwartende Spendenhöhe eines Spenders separat voneinander abgeschätzt, um anschließend die gesamte Spendenhöhe mit beiden Ergebnissen zu ermitteln. Beide Abschätzungen wurden mit *Multi-Layer Perceptrons* (MLP) ermittelt:

1. Bei dem Spendenbetragsmodell wurden nur Spender berücksichtigt, die wirklich gespendet haben (ca. 5% der Daten). Die Ermittlung dieser Eingaben erfolgte durch Training eines Entscheidungsbaums. Dabei wurden alle Attribute, die nicht in diesem Baum vorkamen, ausgelassen. Insgesamt dienten fünf Attribute als Eingabe; zwei aus den ursprünglich 481 Attributen, sowie drei neu generierte.

2. Bei dem Spendenwahrscheinlichkeitsmodell wurde der gesamte Datensatz benutzt. Attribute waren Zeitraum bis zur letzten erfolgten Spende, die Spendenhäufigkeit und Betrag, sowie einige demographische Charakteristika. Zusätzlich wurde das in der Datenexploration für wichtig erachtete Attribut *CONTROLN* einbezogen. Auch mit diesen Attributen wurde ein Entscheidungsbaum gelernt und eine Feature Selection betrieben. Als Eingabe dienten schließlich acht Attribute:

   - vier aus den ursprünglich 481 Attributen
   - die Ausgabe aus dem ersten Schritt
   - zwei für den ersten Schritt generierte Attribute
   - sowie ein Attribut, das für den zweiten Schritt erstellt wurde.

Mit diesem Modell ließ sich ein Spendenbetrag von $ 14877.77 erreichen. Dies sind $ 165.53 mehr, als der Erstplazierte vorhersagen konnte.

14.1.4 Visualisierungen der Daten

Die in dem Artikel erwähnten Unstimmigkeiten in den Daten wurden bei der Nachmodellierung nachvollzogen und zum besseren Verständnis mit *R* neu visualisiert.

Zum einen konnte die unterschiedliche Verteilung der Geburtenjahrgänge gezeigt werden, was auf Probleme bzgl. der Datenintegrität hinweist. (vgl. Abbildung 14.1)

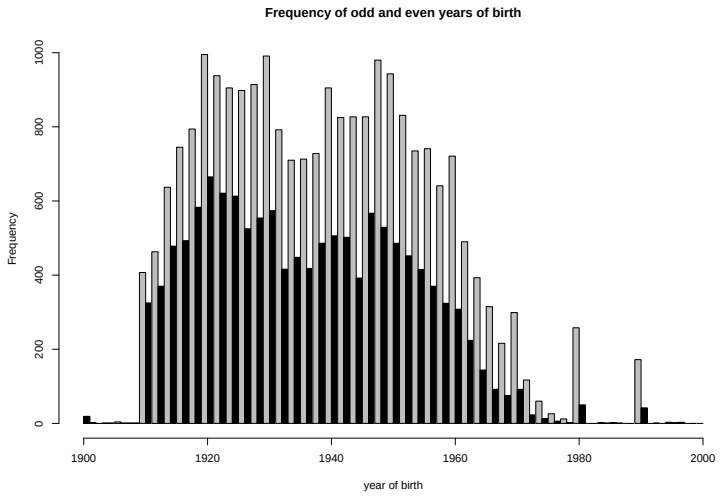


Abbildung 14.1: Verteilung der Geburtsjahrgänge

Auch die scheinbar nicht sequenzielle Vergabe der *CONTROLN* konnte mit Hilfe einer Gegenüberstellung zu den eingenommenen Spenden in der *96NK*- und *96TK*-Kampagne nachvollzogen werden. (vgl. Abbildungen 14.2 und 14.3)

Es zeigte sich ebenfalls ein Zusammenhang zwischen *CONTROLN* und *NUMPROM* bzw. *NUMPROM12* (vgl. Abbildung 14.1.4)

Die Abbildungen zeigen, dass die Kontrollnummer nicht nur eine ID-Funktion besitzt, sondern evtl. einen größeren Anteil für die Vorhersage des Zielattributs leisten kann.

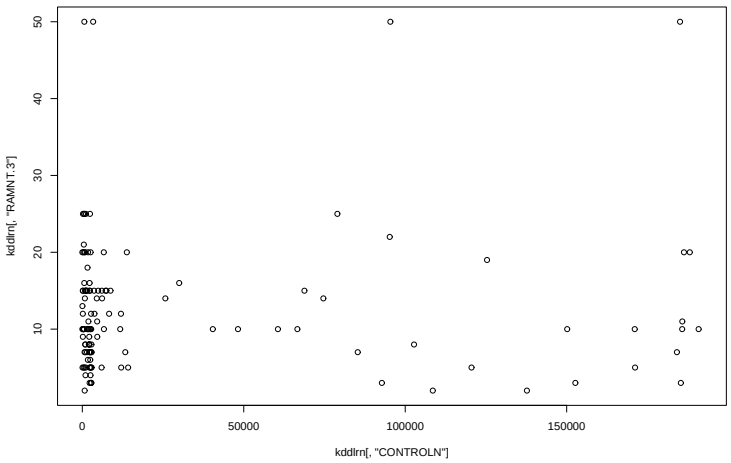


Abbildung 14.2: Spendenbeträge der 96NK-Kampagne vs. *CONTROLN*

14.1.5 Nachmodellierung

Die in der Lösung angesprochenen neuronalen Netze ließen sich bei der Nachmodellierung nicht ohne weiteres übernehmen, da nicht weiter beschriebene Algorithmen, wie z.B. der double-dogleg Algorithmus, verwendet wurden.

Bei dem Versuch, die neuronalen Netze mit SNNS nachzubilden, wurde nur ein unzureichendes Ergebnis erzielt. Aus diesem Grund wurden bei der Nachmodellierung bei der Abschätzung der Spendenwahrscheinlichkeit und des Spendenbetrags *Support-Vector-Machines* verwendet (siehe nächstes Kapitel).

14.2 Praktische Umsetzung

14.2.1 Einführung zur Vorverarbeitung

Entsprechend der schon erwähnten Vorverarbeitung der Daten haben wir einen MiningMart-Fall erstellt, der die Daten so manipuliert, dass man sie direkt in YALE einbinden kann.

Obwohl es sich um einen MiningMart-Fall handelt, werden hier zwei verschiedene Datenmengen bearbeitet. Ein Teil der Operatoren bearbeitet die Testdaten, wohingegen die anderen Operatoren des Falles die Validierungsdaten manipu-

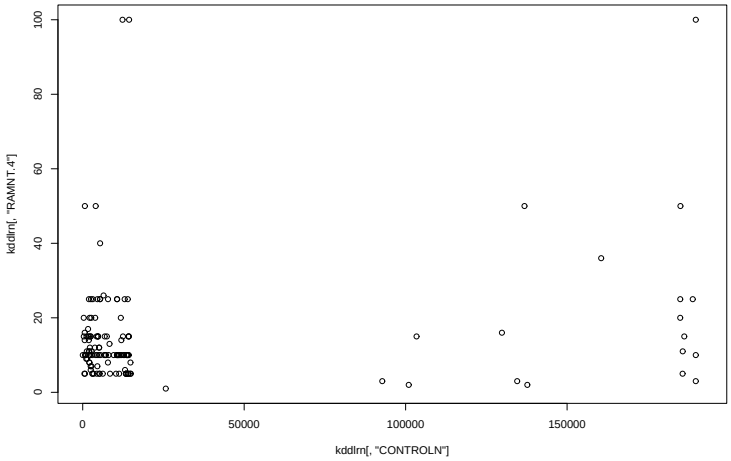


Abbildung 14.3: Spendenbeträge der 96TK-Kampagne vs. *CONTROLN*

lieren.

Da in der Vorverarbeitung aus einer Menge von Datenbankspalten nur ein paar wenige extrahiert werden, wobei einige aus durch mathematische Verknüpfungen einzelner Spalten entstehen, ist MiningMart für diesen Schritt der Analyse-Kette bestens geeignet.

MiningMart kann direkt auf die Datenbank zugreifen und erstellt mithilfe des Operators „New Feature Constructor“ mit Leichtigkeit neue Attribute für die Ziel-View, auf die man dann sql-konform zugreifen kann.

In den theoretischen Ausführungen wurde schon erwähnt, dass für Schritt 1 nur noch die Attribute AVGGIFT, LASTGIFT, AMPERGFT, PGIFTH und MYAMNT für die weiteren Schritte der Analysekette notwendig sind. AMPERGFT, PGIFTH und MYAMNT sind hierbei „erzeugte“ Attribute, die sich wie folgt ergeben:
AMPERGFT ist der durchschnittliche Spendenbetrag von 94NK bis 96NK
PGIFTH ist der Durchschnitt von NGIFTALL pro NUMPROM
MYAMNT ist die Summe von RAMNT_8, 9, 12 und 14

Für Schritt 2 werden nur die Attribute LASTDATE, FISTDATE, INCOME, CONTROLN, PGIFTH, MYAMNT, P_TARGET und SUSPECT benötigt. Erzeugt worden sind PGIFTH, MYAMNT, P_TARGET und SUSPECT, wobei nur noch P_TARGET und SUSPECT erklärt werden müssen:
P_TARGET ist der vorhergesagte Spendenbetrag und kommt aus Schritt 1.
SUSPECT steht für den Durchschnitt der Spenden pro Werbeaktion von Juni 96 bis Juni 97.

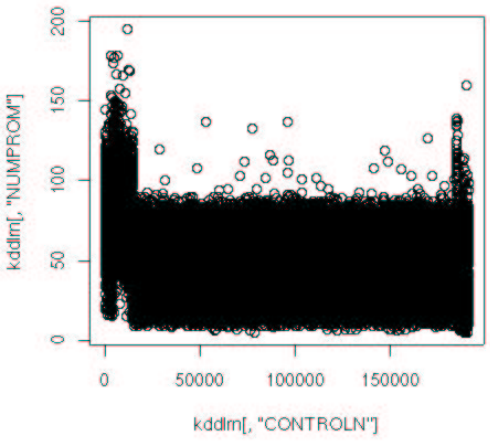


Abbildung 14.4: *CONTROLN* vs. *NUMPROM*

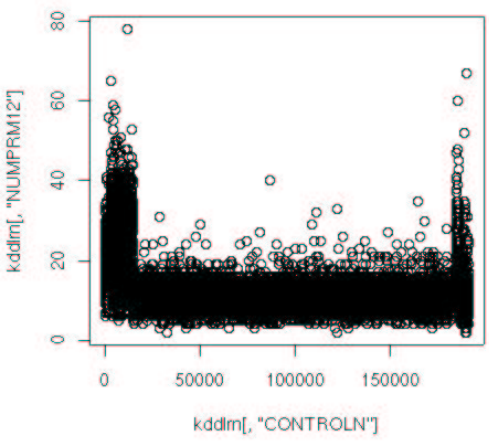


Abbildung 14.5: *CONTROLN* vs. *NUMPROM12*

Bei der Beschreibung des Vorgehens in MiningMart beschränken wir uns nun auf die Erläuterung des Bearbeitens der Testdaten. Die Bearbeitung der Validierungsdaten verläuft natürlich analog.

YALE wird die mit MiningMart erzeugte View dann aufgreifen und die Güte unserer Lösung dann in zwei Schritten berechnen.

14.2.2 Vorverarbeitung mit MiningMart

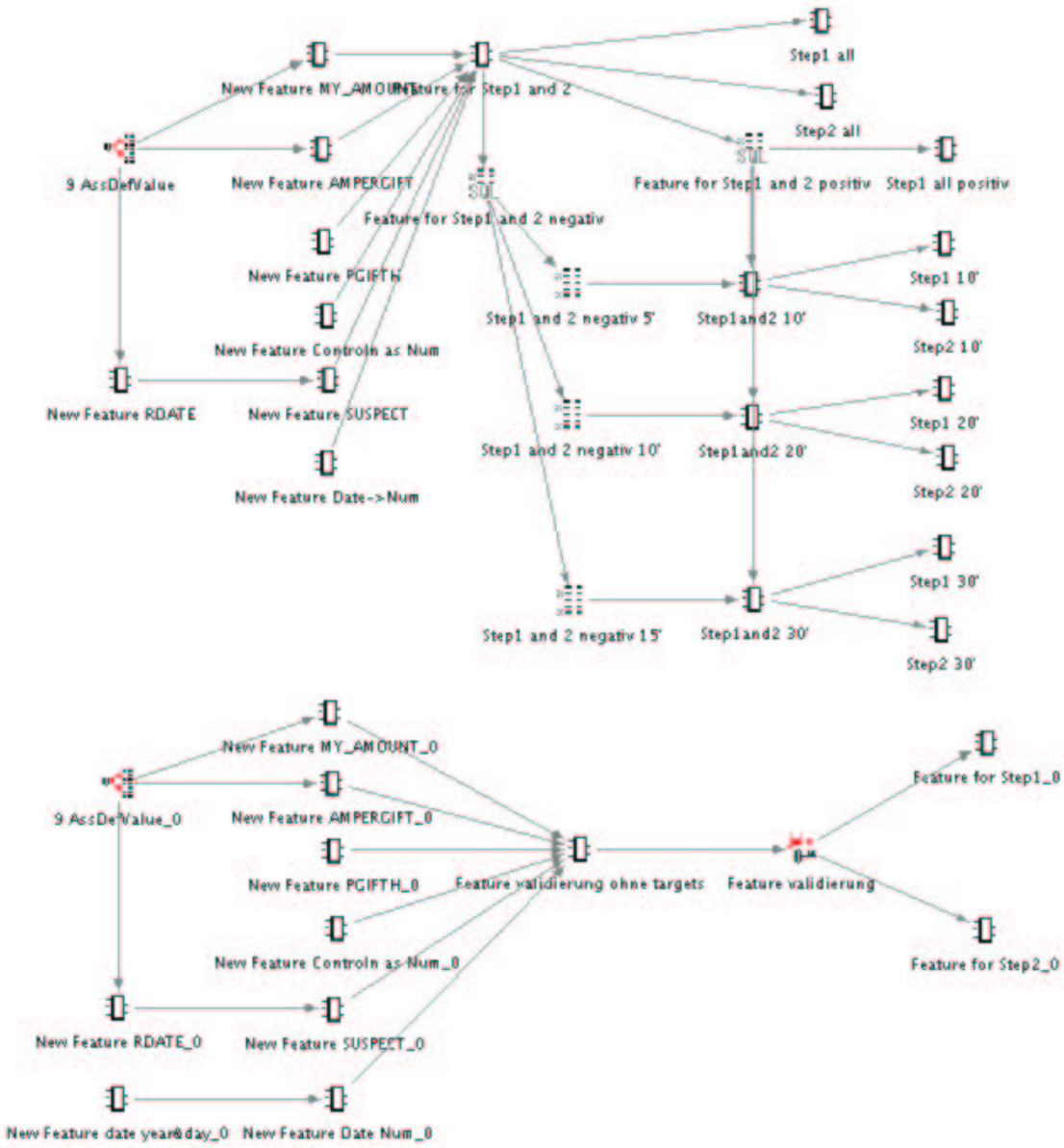


Abbildung 14.6: Verarbeitung mit MiningMart

Wir beschränken uns in diesem Punkt auf der tabellarischen Übersicht der Operatoren unseres Falls, da bereits in der theoretischen Ausführung darauf eingegangen wurde, warum welche Attribute benutzt beziehungsweise verworfen werden.

9 AssDefValue dient dazu, bei numerischen Operatoren null-Values durch

„echte“ Nullen zu ersetzen. Dies ist wichtig, damit bei der mathematischen Verarbeitung der Attribute keine Fehler auftreten.
New Feature RDATE berechnet bei wie vielen Werbeaktionen die aktuelle Person gespendet hat (Hilfsvariable für SUSPECT).
New Feature MY_AMOUNT MY_AMOUNT wird berechnet.
New Feature AMPERGIFT AMPERGFT wird berechnet.
New Feature PGIFTH PGIFTH wird berechnet.
New Feature SUSPECT SUSPECT wird berechnet. Geschätzt wird hier das Verhältnis der Spendenaufforderungen zu den Spenden von Juni 1996 bis Juni 1997
New Feature Date->Num wandelt Datum in Anzahl Monate um (über die Formel Jahreszahl $\times$ 12 + Monate)
New Feature Controln as Num wandelt Controln in numerischen Wert um.
Feature for Step 1 and 2 enthält schliesslich den vorverarbeiteten Datensatz mit den Attributen für Schritt 1 und 2.
Nun erfolgt die Aufteilunng der Daten in positive (Spender) und negative Datensätze. Auf den negativen werden sample der Größe 5000,10000 und 15 Datensätze gezogen (**Step1 and 2 negativ 5', 10', 15'**). Es werden die postiven Daten wieder hinzugefügt.
Nun erfolgt eine Attributauswahl für jedes Sample für Schritt1 und 2 (**Step1 10'** bis **Step2 30'**).

Um den vorverarbeiteten Datensatz zu erhalten, muss man einfach alle Schritte nacheinander kompilieren.

14.2.3 Analysekette in YALE

Die Yale-Kette (vgl. Abbildung 14.7) beginnt natürlich mit einem Root-Element. Danach folgt das Laden der Daten. Mithilfe einen DatabaseExampleSource-Operators wird die von MiningMart erzeugte View (Sample für Schritt 1) geladen.

Über eine Kreuzvalidierung wird nun eine SVM trainiert. Das resultierende Model-File wird gesichert, so dass es kurz darauf auf den Rest der Daten (alle Daten für Schritt 1) angewandt werden kann.

Das Lernergebnis wird nun abgespeichert. Man könnte sagen, dass hier der erste Schritt der Analysekette endet. Für Schritt 2 benötigt man jedoch einige Attribute aus Schritt 1.

Daher wird die Datenbank-View (alle Daten für Schritt 2) geladen und durch Anpassung der XML-Datei mit den Daten aus Schritt 1 verknüpft.

Per Kreuzvalidierung und SVM wird wieder ein Model erstellt und gespeichert.

Das Model wird nun auf alle Daten aus Schritt 2 angewandt. Das Ergebnis wird an einen weiteren CommandLine-Operator übergeben, der ein Programm aufruft, dass die Güte unseres Ergebnisses bestimmt. Die Analysekette ist vorzufinden im Verzeichnis `/pg445/share/ketten/cup98`.

14.3 Lernergebnisse

Das Spendenwahrscheinlichkeitsmodell in Step 2 trennt die Spender in wahrscheinliche und weniger wahrscheinliche Spender. Dabei sind wir so vorgegangen, daß wir für die bekannten Nicht-Spender und Spender jeweils eine Regression mittels einer SVM durchgeführt haben (Yale: libSVM). Die Ergebnisse sind in Abbildung 14.8 zu sehen. Es ist ganz klar zu sehen, daß die Mehrheit der Spender in Graph (a) mehr zur Wahrscheinlichkeit 1 tendiert, wogegen in Graph (b) die Mehrheit zu 0 tendiert.

Die Spendenbeträge werden in Step 1 vorhergesagt. Abbildung 14.9 zeigt die zu erwartenden Spendenbeträge für zuvor bekannte Spendenbeträge auf.

Die Güte wird dann mittels eines Java-Programmes ermittelt. Bei mehreren Läufen mit verschiedenen Parametern wurde auf dem Lerndatensatz eine Güte von $ 10.681,- erreicht! Auf dem Validierungsdatensatz hingegen nur $ 9230,-. Diese Güte würde die 17. Platzierung bedeuten.

Ein Grund dieses schlechten Ergebnisses entsteht hauptsächlich durch Overfitting im Schritt 2. Da erstellt Modell der Support Vector Machine enthält in der Menge immer ungefähr 80% der Datensatzmenge an Stützvektoren. Diese Problematik des Overfitting wird auch in dem Paper angesprochen, auf dem wir unsere Modellierung aufgebaut haben. Um Overfitting zu vermeiden, wurde das Neuronale Netz mit Gewichtsdämpfung trainiert und das Training frühzeitig abgebrochen.

Eine weitere Ursache kann die gewählte Datenrepräsentation für die Datumswerte sein, da wir durch Visualisierungen feststellen konnten, daß die Daten sehr schlecht voneinander zu trennen sind. Für die Repräsentation der Datumswerte sind im Paper keinerlei Angaben zu finden.

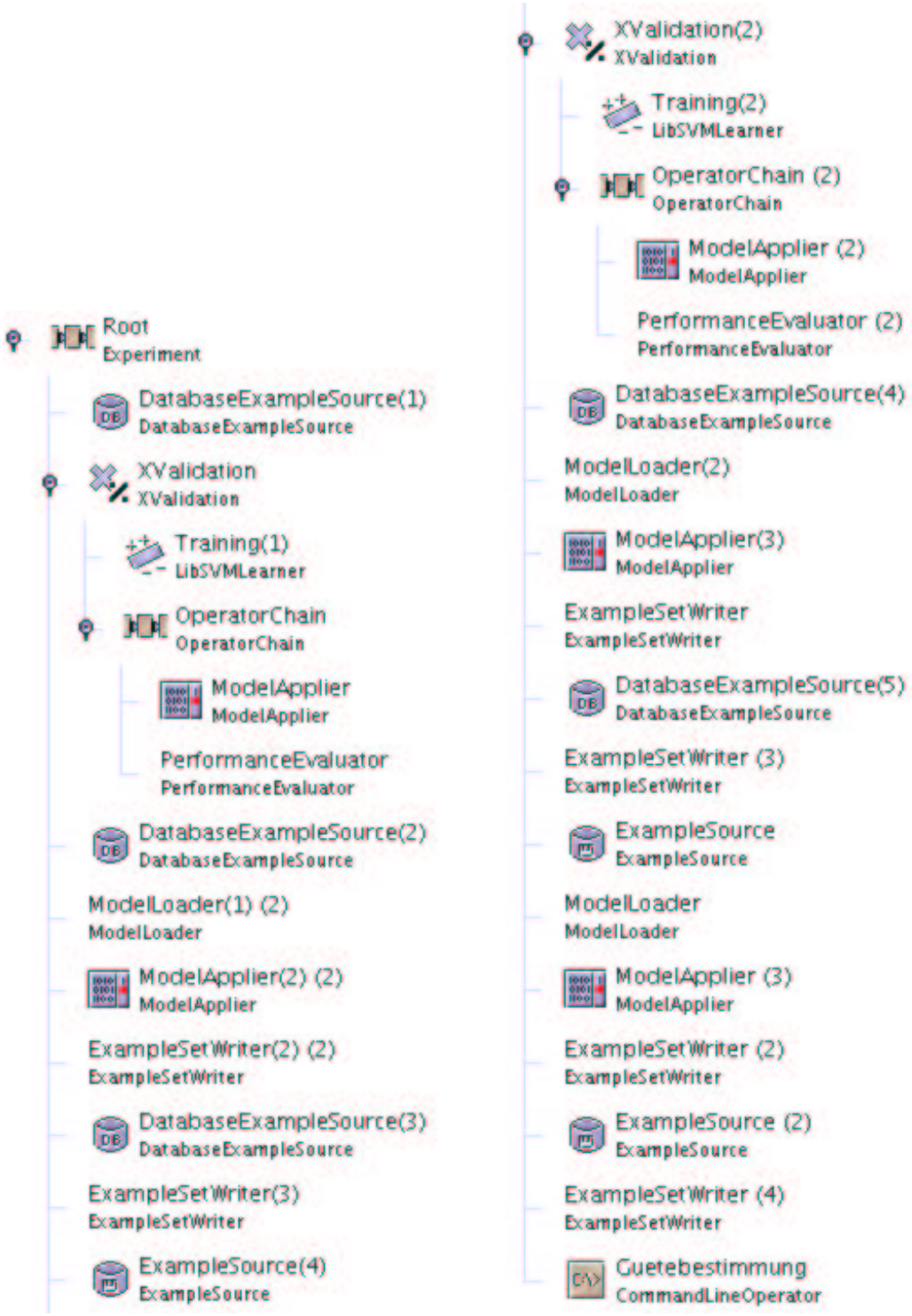


Abbildung 14.7: Verarbeitung mit YALE

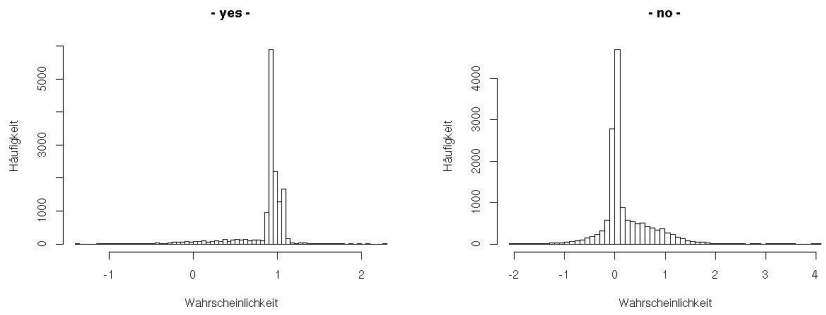


Abbildung 14.8: Spendenwahrscheinlichkeitsmodell (Step 2); (a) zeigt die durch die Regression ermittelte Häufigkeit der Spendenwahrscheinlichkeit für bekannte, zukünftige Spender (b) zeigt die Häufigkeit der Spendenwahrscheinlichkeit für bekannte, zukünfitge Nicht-Spender

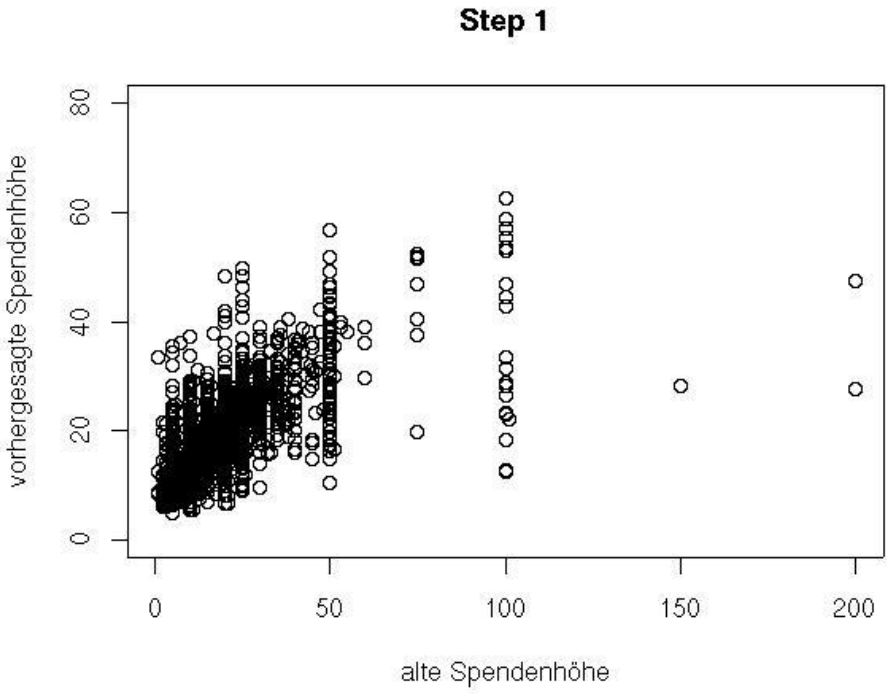


Abbildung 14.9: Spendenbetragsmodell

Kapitel 15

KDD-Cup 1999

Marcel Gaspar, Marc Twiehaus, Nazif Veliu

15.1 Die Aufgabe

Die Aufgabe des KDD-Cup 99 bestand darin, einen Klassifizierer zu entwickeln, der zwischen legalen (sog. normalen) und illegalen (sog. Angriffen) Verbindungen zu einem Computer-Netzwerk unterscheiden kann. Hierzu wurde von den MIT Lincoln Labs eine simulierte Umgebung eines typischen U.S. Air Force LANs geschaffen, welches für 9 Wochen in Betrieb genommen wurde und dabei zahlreichen Angriffen ausgesetzt war. Aus diesen Daten wurde dann das Trainings- und das Test-Set erstellt, wobei das Trainings-Set aus ungefähr 5 Millionen und das Test-Set aus ungefähr 300.000 Verbindungen bestand. Eine Verbindung ist dabei eine Sequenz von TCP-Packeten, die durch 41 Attribute beschrieben werden, und die Attribute sind numerisch, nominal und boolesch. Jede Verbindung ist zusätzlich mit einem Ziel-Label gekennzeichnet, das entweder mit "normal" oder genau einem Angriffstyp bezeichnet ist. Die Angriffstypen teilen sich dabei in 4 Ankgriffskategorien auf:

- **DOS**: denial of service, z.B. syn flood
- **R2L**: (remote to local) unauthorisierter Zugriff von einem äußeren Computer, z.B. Password raten

- **U2R**: (user to root) unauthorisierter Zugriff auf lokalen Superuser Rechte, z.B. buffer overflow
- **PROBE**: (probing) Überwachungen und andere "Spionage"-Aktivitäten, z.B. port scanning

Von besonderer Wichtigkeit ist hierbei, daß das Test-Set nicht die gleiche Wahrscheinlichkeitsverteilung des Ziel-Labels wie das Trainings-Set besitzt, und außerdem Angriffstypen enthält die im Trainings-Set nicht vorkommen. Hiermit sollte die Aufgabe realistischer gemacht werden. Netzwerk-Spezialisten vermuten nämlich, dass die meisten neuen Angriffstypen nur Varianten bereits bekannter Angriffstypen sind und das die Signatur bekannter Angriffe ausreicht um neue Varianten erkennen zu können. Insgesamt kamen 38 Angriffstypen vor, 24 im Trainings-Set und 14 zusätzliche im Test-Set. Im Detail sehen die Verteilungen dabei wie folgt aus:

	Training	Test
normal	19,69 %	19,48 %
probe	0,83 %	1,34 %
dos	79,24 %	73,90 %
u2r	0,01 %	0,07 %
r2l	0,23 %	5,20 %

Tabelle 15.1: Verteilung der Labels

Die u2r und r2l Angriffe machen also zusammen 5,27 % des Test-Set aus, ein deutlicher Anstieg gegenüber den 0,24 % des Trainings-Set. Jede Abgabe wurde durch die folgende Kostenmatrix bewertet:

	normal	probe	dos	u2r	r2l
normal	0	1	2	2	2
probe	1	0	2	2	2
dos	2	1	0	2	2
u2r	3	2	2	0	2
r2l	4	2	2	2	0

Tabelle 15.2: Die Strafkosten

Die durchschnittlichen Kosten die entstehen, wenn man alle Beispiele des Test-Sets als "probe" klassifiziert, betragen übrigens 0,5220 Punkte, was immer noch besser ist als die durchschnittlichen Kosten der Abgabe des letzten Platzes.

15.1.1 Die Gewinner-Lösung

Die Gewinner-Lösung wurde von Dr. Bernhard Pfahringer vom östereichischen Institut für künstliche Intelligenz eingereicht und verursachte durchschnittliche Strafkosten von 0,2331 Punkten pro Vorhersage. Hier die dazugehörige Konfusionsmatrix:
Hierbei bedeutet der obere linke Eintrag, dass 60262 der tatsächlich "norma-

	predicted	normal	probe	dos	u2r	r2l	% correct
actual							
normal		60262	243	78	4	6	99,5 %
probe		511	3471	184	0	0	83,3 %
dos		5299	1328	223226	0	0	97,1 %
u2r		168	20	0	30	10	13,2 %
r2l		14527	294	0	8	1360	8,4 %
% correct		74,6 %	64,8 %	99,9 %	71,4 %	98,8 %	

Tabelle 15.3: Konfusionsmatrix der Gewinnerlösung

len" Beispiele des Test-Sets auch als "normal" vorhergesagt wurden. Der obere Eintrag der letzten Spalte besagt, dass 99,5% der als "normal" vorhergesagten Beispiele auch tatsächlich "normal" waren, und der erste Eintrag der letzten Zeile besagt schließlich, dass 74,6% der "normalen" Beispiele des Test-Set als "normal" vorhergesagt wurden.

Der benutzte Ansatz ist der eines kosten-sensitiven bagged boosting11, wobei ein Ensemble von 50 mal 10 Entscheidungsbäumen konstruiert wurde:

- 50 Samples wurden aus den 5 Millionen Ausgangsbeispielen des Trainingsdatensatzes gezogen. Im Gegensatz zum normalen bagging11 wurde das Sample jedoch disproportional geschichtet gezogen, so dass immer alle Beispiele für die beiden am wenigstens vertretenen Klassen (u2r,r2l), 4000 Beispiele für "probe", 80.000 für "normal" und 400.000 für "dos" in einem Sample enthalten waren. Zusätzlich wurden die doppelten Einträge entfernt.
- Für jedes Sample wurde ein Ensemble von 10 C5 Entscheidungsbäumen geschaffen, welches sowohl C5s Optionen für Strafkosten als auch für das bagging berücksichtigte.
- Die finalen Vorhersagen wurden auf den 50 Einzel-Vorhersagen der Sub-Ensembles unter Minimierung des sog. "conditional risk" (Summe der

Strafkosten mal Klassen-Wahrscheinlichkeit) berechnet.

15.2 Eigene Lösung - Beschreibung der Operatorketten

Grundidee:

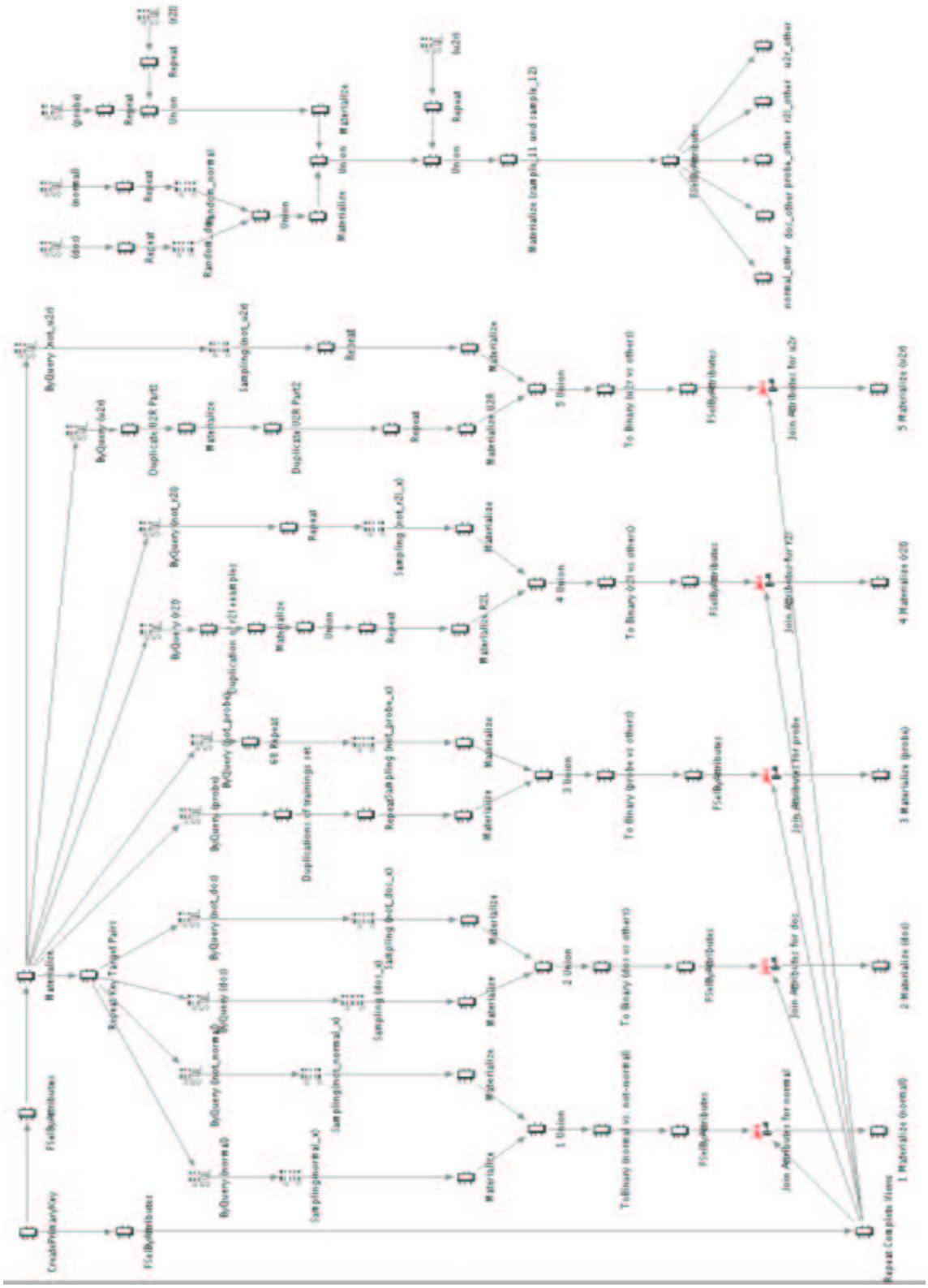
1. Aufteilung des 5-Klassen-Problems in 5 2-Klassen-Probleme (jeweils 1 Klasse gegen alle anderen)
2. Für jedes der 5 2-Klassen-Probleme 10 Modelle erstellen
3. Auf diesen 10 Modellen jeweils durch Meta-Lernen ein Meta-Modell für jede Klasse erstellen, wobei ein unabhängiges Sample zum Lernen benutzt wird.
4. Auf diesen 5 Meta-Modellen durch Meta-Meta-Lernen ein finales Modell erstellen, wobei ein weiteres unabhängiges Sample zum Lernen benutzt wird

15.2.1 Mining Mart

In Mining Mart werden mittels verschiedener Operatorketten die für das Meta-Lernen benötigten Samples erstellt. Ziel ist es zunächst, dass 5-Klassen Problem in 5 2-Klasssen Probleme zu unterteilen, also jede Klasse gegen alle anderen Klassen abzuwägen. Zur Verbesserung des Lernergebnisses werden alle Samples disproportional geschichtet nach ihren Klassen gezogen.
Im einzelnen sind dies:

- 5 x 10 Samples der Größe 100.000 für die Modell-Erzeugung, wobei die eine Hälfte des Samples aus Instanzen der aktuell zu lernenden Klasse besteht, und die andere Hälfte aus zufällig gezogen Instanzen der 4 anderen Klassen. Für die kleinen Klassen (u2r, r2l und probe) wurden die vorhandenen Instanzen vervielfältigt um die obige Bedingung zu erfüllen.
- 1 Sample der Größe 300.000 für das Meta-Lernen auf den 10 2-Klassen-Modellen. Dieses Sample liefert auch die Kontroll-Spalte für das im nächsten Schritt ausgeführte Meta-Lernen auf den 2-Klassen-Modellen

- 1 Sample der Größe 300.000 für das Meta-Lernen der 5 2-Klassen-Meta-Modelle: Dieses Sample liefert auch die Kontroll-Spalte für das im nächsten Schritt ausgeführte Meta-Lernen auf den 2-Klassen-Meta-Modellen



Sampling Kette in Mining Mart

15.2.2 Modellierung in Yale - Vorüberlegung

Für die Operationalisierung des modellierten Falles haben wir folgende Einteilung der verwendeten Operationen in (Meta-)Ebenen vorgenommen:

1. Level 0: Operationen auf den Ausgangsdaten
   (Ausgangsdaten = 41 Attribute $\in \{duration, service, flag, ...\}$ + Label $\in \{normal, dos, probe, u2r, r2l\}$)
2. Level 1: Operationen auf den Zweiklassenvorhersagen
   (Zweiklassenvorhersagen = 10 Attribute $\in \{PredM1, PredM2, ..., PredM10\}$ + Label $\in \{normal, other\} \vee \{dos, other\} \vee ...$)
3. Level 2: Operationen auf den Fünfklassenvorhersagen
   (Fünfklassenvorhersagen = 5 Attribute $\in \{PredM1, PredM2, ..., PredM5\}$ + Label $\in \{normal, dos, probe, u2r, r2l\}$)

In Yale wurden insgesamt 5 Operatorketten angelegt, 3 für das Lernen und 2 für die Vorhersage auf dem Testset. Die Ketten werden im Folgenden in der Reihenfolge ihrer Anwendung vorgestellt.

15.2.3 Lernen der Modelle Level0 bis Level2

ModellLernenLevel0

erstellt auf Samples der Ausgangsdaten für jede Klasse 10 Zweiklassen Basis-Modelle und wendet diese auf ein neues Sample an
einmal pro Klasse ausführen, wobei das neue Sample zur Modellanwendung in jedem Durchlauf dasselbe ist

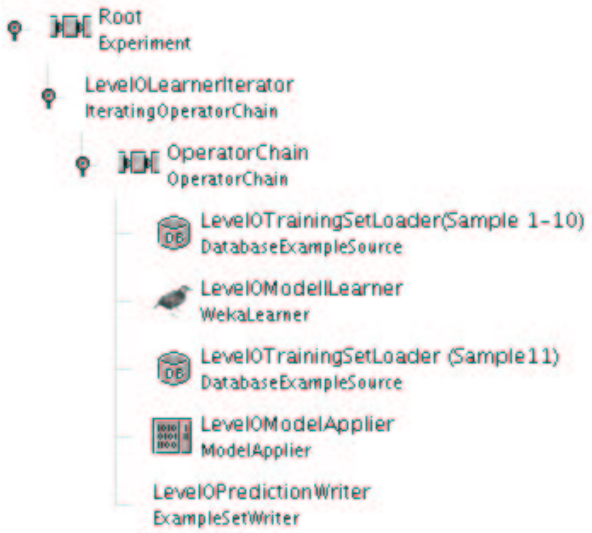
Eingabe:

- jeweils 10 Samples aus TrainingSetLevel0 (Ausgangsdaten - aus Mining-Mart) für den Level0-Lerner (AdaBoostM1)

- 1 unabhängiges Sample ("Sample11") aus TrainingSetLevel0 für Level0-Modell-Anwendung

Ausgabe:

- jeweils 10 auf den Ausgangsdaten gelernte (2Klassen-)Modelle Level0
- jeweils 10 Vorhersagen Level0 der erstellten Modelle auf Sample11 (die Vereinigung aller Vorhersagen wird dem TrainingSetLevel1 für die folgende Kette entsprechen)



Kette Modell Lernen Level0

Für den nächsten Schritt wird für jede Klasse das dem entsprechendem Zweiklassenproblem angepasste Ziellabel des Sample11 benötigt -aus MiningMart (für den Metalernschritt z.B. der Klasse "normal" müssen alle von "normal" verschiedenen Werte nach "other" umbenannt werden.

ModellLernenLevel1

erstellt auf den Zweiklassenvorhersagen des vorherigen Schrittes für jede Klasse 1 Zweiklassen Meta-Modell und wendet dieses auf ein neues Sample an
einmal pro Klasse ausführen, wobei das neue Sample zur Modellanwendung in

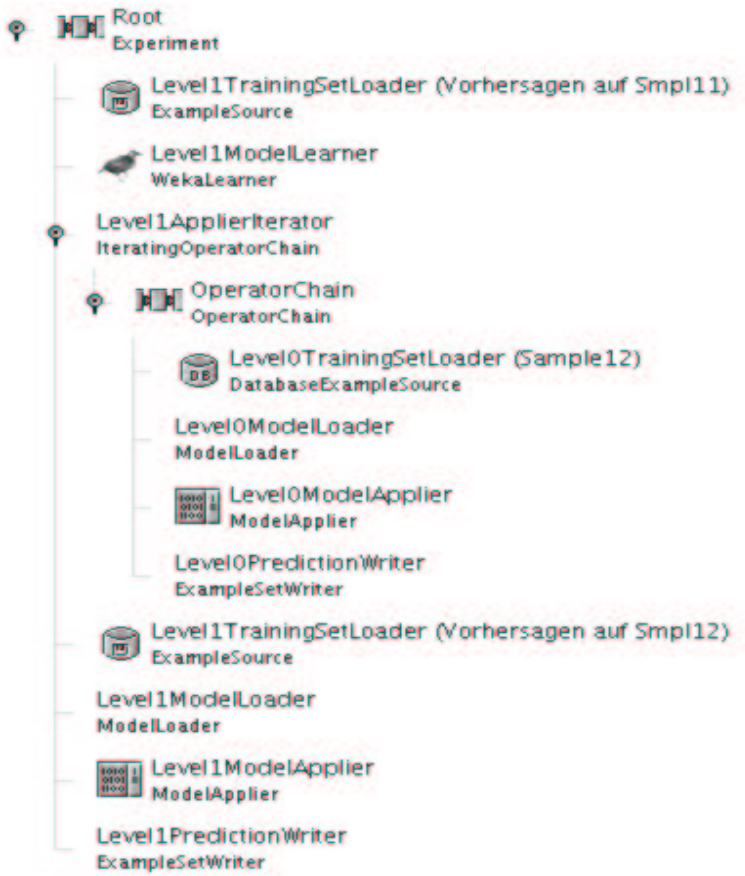
jedem Durchlauf dasselbe ist

Eingabe:

- jeweils die 10 Modelle Level0 aus vorherigem Schritt
- das angepasste Ziellabel aus Sample11
- jeweils die 10 Vorhersagen Level0 auf Sample11 aus der vorherigen Kette für den Level1-Lerner (J48 unpruned)
  Achtung: An dieser und allen folgenden Stellen, an denen die Vorhersagen eines niedrigeren Levels durch das Einladen vereinigt werden zu einem TrainingSet höheren Levels, muss in einer zuvor angelegten xml-Datei auf die Vorhersagen und das Ziellabel aus dem jeweils vorherigen Level in den Attribut-Tags der xml-Datei mittels Parameter "sourcefile" verwiesen werden (siehe Yale-Manual). Auf diese Weise wird mit Hilfe des Operators ExampleSource der Einsatz des ExamplesourceMergers umgangen, welcher aufgrund quadratischer Laufzeit zu uneffizient auf großen Datenmengen ist.
- 1 weiteres unabhängiges Sample ("Sample12") aus TrainingSetLevel0 für Level1-Modell-Anwendung

Ausgabe:

- jeweils ein (2Klassen-)Modell Level1
- jeweils 1 Vorhersage Level1 der erstellten Modelle auf Sample 12 (die Vereinigung aller Vorhersagen wird dem TrainingSetLevel2 für die folgende Kette entsprechen)



Kette Modell Lernen Level1

Für den nächsten Schritt wird das Ziellabel des Sample12 benötigt -aus MiningMart

ModellLernenLevel2

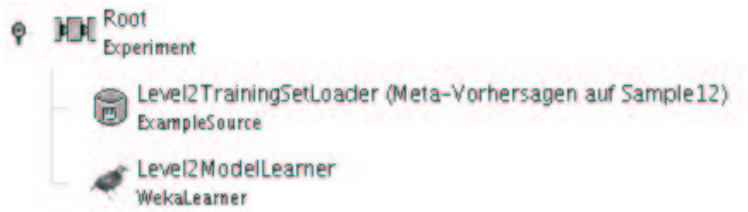
erstellt auf den Zweiklassen Meta-Vorhersagen des vorherigen Schrittes 1 Fünfklassen Meta-Meta-Modell
einmal ausführen
Eingabe:

- die 5 Modelle aus der 5mal wiederholten vorherigen Kette
- das Ziellabel aus Sample12
- die 5 Vorhersagen Level1 auf Sample12 aus der 5 mal wiederholten vor-

herigen Kette für den Level2-Lerner (J48 unpruned)

Ausgabe:

- ein (5Klassen-)Modell Level2



Kette Modell Lernen Level2

15.2.4 Vorhersage auf Testdaten

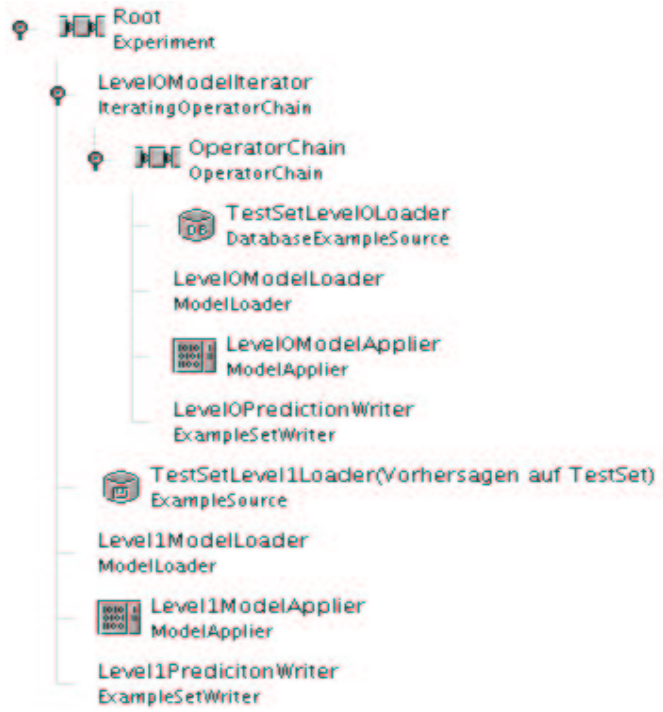
VorhersageAufTestSetLevel0+1

wendet die Modelle Level0 und Level1 auf einen Testdatensatz an
einmal pro Klasse ausführen
Eingabe:

- das TestSet (entspricht Level0)
- jeweils die 10 Modelle Level0 aus ModellLernenLevel0
- jeweils 1 Modell Level1 aus ModellLernenLevel1

Ausgabe:

- jeweils 1 (2Klassen-)Vorhersage Level1 auf dem TestSet



Kette Vorhersage auf Testset Level0 und 1

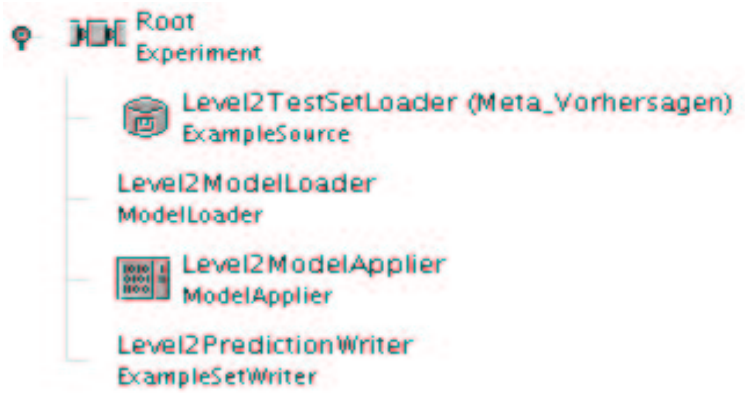
VorhersageAufTestSetLevel2

wendet das Modell Level2 auf die Ausgabe des vorherigen Schrittes an
einmal ausführen
Eingabe:

- die 5 Vorhersagen aus der 5 mal wiederholten vorherigen Kette
- das Modell Level2 aus ModellLernenLevel2

Ausgabe:

- (5Klassen-)Vorhersage auf TestSet



Kette Vorhersage auf Testset Level2

15.2.5 Bewertung der Ergebnisse

Für die Bewertung der Ergebnisse liegen die beiden von den Cup99-Veranstalltern verwendeten awk-Scripte "categorization.awk" und "scoring.awk" bereit. Die Ausgabe der Konfusionssmatrix zu z.B. "vorhersage.data" kann durch den Aufruf "awk -f categorization.awk vorhersage.data | awk -f scoring.awk" erfolgen. Mit den hier beschriebenen Einstellungen lieferten unsere Ketten die folgende Matrix. Die durschnittlichen Kosten pro Vorhersage, welche ja als Bewer-

	predicted	normal	probe	dos	u2r	r2l	% correct
actual							
normal		59513	390	687	2	1	98,2 %
probe		160	3763	243	0	0	90.3 %
dos		5193	1256	223404	0	0	97.2 %
u2r		54	133	59	0	0	0.0 %
r2l		15057	515	577	21	1	0,0 %
% correct		74.4 %	62.1 %	99.3 %	0.0 %	50.0 %	

Tabelle 15.4: Konfusionsmatrix der Eigenen Lösung

tungsgrundlage der Lösung dienten, betragen 0.2477 und das entspricht dem 8. Platz.

15.2.6 Ein alternativer Ansatz

Die Gruppe hat sich nebenher auch mit alternativen Ansätzen beschäftigt und versucht, die Strafkosten mit in den Lernprozeß einzubringen. Zwar konnte mit diesem Ansatz der 1. Platz geschlagen werden, da hierbei aber eher Glück im Spiel war, soll dies hier nur kurz erwähnt werden. Als Trainingsdatensatz wurde der offiziele 10 Prozent Trainingsdatensatz verwendet, wobei die drei am wenigsten vertretenen Klassen, also "probe", "u2r" und "r2l", verdreifacht wurden. Insgesamt enthielt das Trainings-Sample 504.590 Instanzen, davon 97.277 "normal", 391.458 "dos", 12.321 "probe", 156 "u2r" und 3.378 "r2l". Als Meta-Lerner diente der Weka-Algorithmus "CostsensitiveClassifier" mit der offiziellen Strafkostenmatrix. Dieser Lerner macht seinen Basisklassifizierer kostensensitiv, und als Basisklassifizierer wurde dann AdaBoostM1 mit 10 Iterationen gewählt. Als Basisklassifizierer für AdaBosot wurde dann wiederum J48 mit den Default-Einstellungen gewählt. Als Test-Satz diente ein Sample mit 297.280 Instanzen, davon 50.811 "normal", 204.189 "dos", 41.102 "probe", 52 "u2r" und 1.126 "r2l". Das gelernte Modell erreicht dabei eine accuracy von 83,01 % und eine precision von 79,45 % . Auf dem offizielen Test-Set, welches als Bewertungsgrundlage für den Cup diente, erreichte es eine durchschnittliche Punktzahl von 0,2327 Punkten pro Vorhersage, und ist damit um 0,0004 Punkte besser als der Gewinner. Die Verbesserung gegenüber der Gewinner-

	predicted	normal	probe	dos	u2r	r2l	% correct
actual							
normal		57631	545	2395	9	13	95,1 %
probe		925	3097	142	1	1	74,3 %
dos		684	683	228386	0	100	99,4 %
u2r		129	85	19	10	3	4,1 %
r2l		15356	208	548	1	58	0,4 %
% correct		77,1 %	67,1 %	98,7 %	47,6 %	33,1 %	

Tabelle 15.5: Konfusionsmatrix der Alternativlösung

Lösung besteht hier aber nur in der besseren Vorhersage für die Klasse "probe". Insbesondere die beiden kleinen Klassen wurden schlechter erkannt. Das Ziel, die beiden kleinen Klassen besser vorherzusagen, wurde also mit diesem Modell nicht erreicht.

Kapitel 16

KDD-Cup 2000

Dirk Dach, Christophe Foussette, Christian Kullmann, Anna Litvina, Lars Michele

16.1 Einleitung

Beim KDD Cup 2000 waren 234954 Sessions eines Onlinehändlers gegeben. Bei Frage 1 sollte eine Vorhersage getroffen werden, ob eine Session fortgeführt wird oder nicht. Somit handelt es bei Frage 1 um eine Klassifikationsaufgabe bei der ein überwachtes Lernverfahren zum Einsatz kommen soll. Anhand eines Testdatensatzes sollte die Güte der Lösung bestimmt werden. Als Gütemaß wurde die Accuracy, die Anzahl der korrekt klassifizierten Sessions, herangezogen. Bevor in den Abschnitten 16.4 und 16.5 die konkreten Analyseketten vorgestellt werden, werden im Abschnitt 16.2 zunächst eine Übersicht über die Daten gegeben und in Abschnitt 16.3 das nachzubildende Modell sowie die Abweichungen davon beschrieben.

16.2 Datenbeschreibung

Die zur Verfügung gestellten Daten repräsentieren 234954 Sessions. Diese Daten wurden aus Clickdaten, d.h. Weblogs gewonnen. Das Zielattribut Session-

Continues ist True, wenn die Session weitergeführt, und ansonsten False. Die 296 Eingangsattribute sind sowohl numerisch als auch nominal und beschreiben eine Session. So wird mit den numerischen Attributen hauptsächlich die Anzahl der Besuche bestimmter Seiten gezählt. Die nominalen Attribute geben zum Beispiel Auskunft darüber, welchen Browser der Benutzer verwendet hat, oder welchen Webserver er vorher besucht hatte. 105 der 296 Eingangsattribute sind personenbezogene Daten, die vom Dienstleister Acxiom hinzugekauft wurden. Da diese Daten nur von bei Acxiom registrierten Kunden zur Verfügung stehen, sind diese Attriubte nur bei ca. 1% der Lerndaten gesetzt.

16.3 Das nachzubildende Modell

In [24] ist das Modell der Gewinnerlösung beschrieben. Dieses Modell ist in Abbildung 16.1 dargestellt. Im ersten Schritt werden die Daten in Oneclick und Mulitclick Sessions getrennt (60% Oneclick Session, 40% Multiclick Sessions), da Oneclick Session eine Besonderheit darstellen. 85% von ihnen kommen durch Bots zustandekommen, die nach einem Click ihre Session beenden (SessionContinues False). Diese Bots wurden im nächsten Schritt gefiltert und alle mit False vorhergesagt. Für verbleibenden Oneclick Sessions wurde durch die Gewinner ein Modell mit fünf Regeln angewendet. Da in [24] nicht näher dokumentiert wurde, wie diese fünf Regeln zustandegekommen sind, werden diese Regeln durch einen Entscheidungsbaum ersetzt. Ca. 28% der Multiclick Sessions sind so genannte Partial Test Patterns. Das sind Sessions, die von Webentwicklern durchgeführt werden, und mit hoher Wahrscheinlichkeit fortgesetzt werden. Daher werden diese direkt mit True vorhergesagt. Für die verbleibenden Multiclick Sessions wurden drei verschiedene Modelle gelernt und diese dann kombiniert. Da aus [24] auch nicht klar hervorgeht, wie diese Modelle entstanden sind, werden hierfür verschieden gestutzte (pruning) Entscheidungsbäume verwendet.

16.4 Analysekette zum Lernen des Modells

Zunächst wurden die Lern- und Testdaten in die Oracle Datenbank FBI als Tabellen kdd2000_q1agg und kdd2000_q1agg_test eingefügt. Die Trennung der Daten wird mit MiningMart durchgeführt, das Lernen der Modelle mit Yale bzw. Weka.

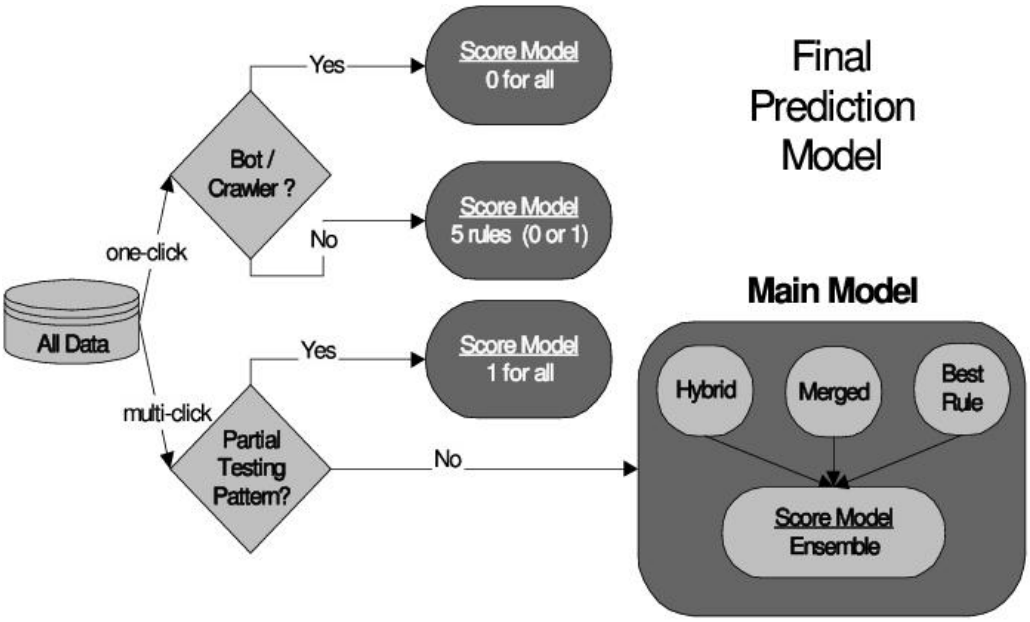


Abbildung 16.1: Schematische Darstellung des Gewinnermodells

16.4.1 Trennen der Daten in One- und Multiclick Sessions

Anhand des Attributs SessionRequestCount werden die Daten in One- und Multiclick Sessions getrennt. Ist SessionRequestCount gleich eins, ist die Session eine Oneclick Session, ansonsten eine Multiclick Session.

16.4.2 Trennen der Oneclick Sessions

Aus den Oneclick Sessions werden mit Hilfe des Attributs SessionUserAgent die Bots herausgefiltert. Eine Session wird als durch einen Bot durchgeführte angesehen, wenn SessionUserAgent einem der folgenden Strings ähnelt:

1. LWP::Simple/5.21
2. DIIbot/1.0
3. Lotus-Notes/4.5 (Windows-NT)
4. Mozilla/3.01 [en] (Win95; I)
5. Mozilla/3.01 (compatible;)
6. Teleport Pro/1.29 via NetCache version 3.1.2d-Solaris
7. Offline Explorer/1.3

8. Mozilla/4.0 (compatible; MSIE 5.01; MSN 2.6; Windows 98; MSIECrawler)
9. NetMechanic V2.0
10. Cute MP3 and File Finder (CuteFTP)
11. Mozilla/3.0 (compatible; PerMan Surfer 3.0; Win95)
12. WebTrends/3.0 (WinNT)
13. Lesszilla/0.01 [en] (platform independent)
14. Mozilla/3.Mozilla/2.01 (Win95; I)
15. Lesszilla/0.02 (Database Link Validator)
16. xyro
17. Microsoft Internet Explorer/4.40.426 (Windows 95)
18. Offline Explorer/1.2
19. xyro_(xcrawler@inria.fr)
20. Mozilla/5.0 (compatible; MSIE 5.0)
21. Mozilla/4.0 (compatible; MSIE 5.0; Win32)
22. Teleport Pro/1.29
23. Mozilla/4.0
24. Mozilla/4.0 (TuringOS; Turing Machine; 0.0)
25. unknown

16.4.3 Trennen der Multiclick Sessions

Um die Partial Test Patterns aus den Multiclick Sessions zu filtern, wird das Attribut SessionFirstReferrer genutzt. Fängt SessionFirstReferrer mit "file://" an, so wird dieses Tupel als Partial Test Pattern identifiziert.

16.4.4 Operatorkette zum Lernen der Oneclick Sessions

Zum Lernen der Oneclick Sessions ohne Bots wird Yale verwendet. In einer OperatorChain wird zunächst über das DBExampleSourceReader die von MiningMart erzeugte View. Das Modell wird mit dem J48 Entscheidungsbaum von Weka gelernt und mit einer 10-fachen Kreuzvalidierung getestet. Die beste Parametereinstellung, d.h. die Parametereinstellung, die die beste Accuracy erzeugt, war für C, der Wert für die Konfidenz einer Regel, 0.95 und für M, die minimale Anzahl von Tupeln in einem Blatt, 2. Das Modell wird gespeichert, um es bei der Vorhersage zu verwenden. Beim Lernen fiel auf, dass gewisse Attribute entweder die Tupel eindeutig identifizieren oder eine ungenügende

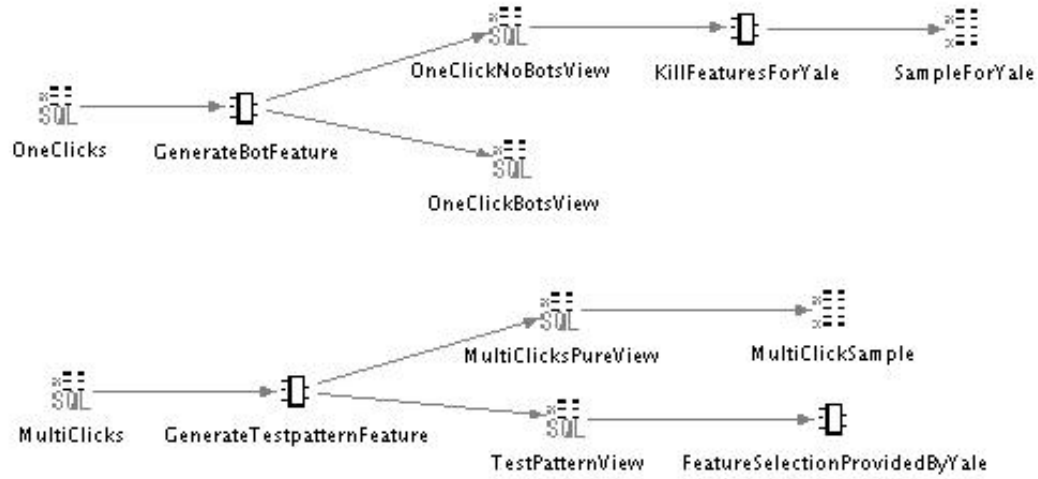


Abbildung 16.2: MiningMart Operatorkette zum Lernen der Trennen der Sessions

Varianz besitzen und somit weggelassen werden müssen. Sie sind im Anhang angegeben.

16.4.5 Operatorkette zum Lernen der Multiclick Sessions

Diese Operatorkette besteht aus drei Teilen.
Im ersten Teil werden die einzelnen Modelle, nach [24] Best-, Merge- und HybridModel benannt, gelernt. Die Multiclick Sessions ohne Partial Test Patterns sind nach der Bestimmung durch MiningMart in einer View gespeichert. Diese View wird mit dem DBExampleSourceReader vor jedem Lernen geladen. 24 Attribute werden beim Lernen weggelassen, da sie die Beispiele eindeutig identifizieren und somit nutzlos für ein Lernverfahren sind, sie sind im Anhang aufgelistet. Das BestModel ist ein stark gestutzter J48 Entscheidungsbaum. Die starke Stutzung wird mit den die Parametereinstellungen C 0,96 und M 10 erreicht. Das MergeModel ist ein J48 Entscheidungsbaum, der die beste Accuracy erreicht hat. Dies geschieht mit den Parametereinstellungen C 0,99 und M 5. Das HybridModel wird durch einen J48 Entscheidungsbaum mit C = 0,95 und M = 5 von Weka gelernt. Die drei Modelle werden mit Yale abgespeichert, um sie im zweiten Teil der Operatorkette zu verwenden. Um eine Abschätzung der Güte (Accuracy) der gelernten Modelle zu erhalten, wird eine zehnfache

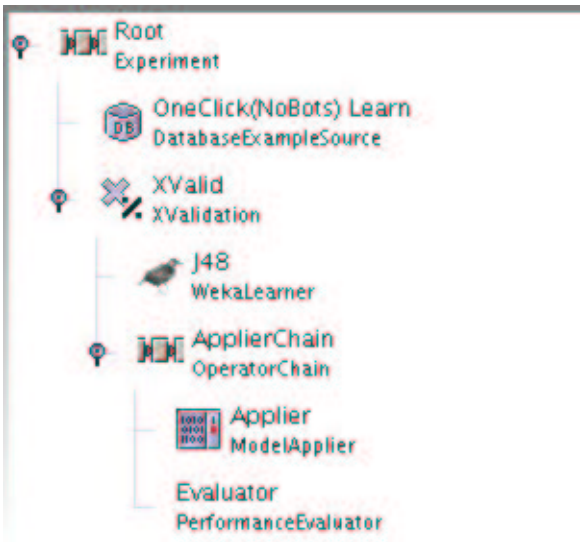


Abbildung 16.3: Yale Operatorkette zum Lernen der Oneclick Sessions

Kreuzvalidierung durchgeführt.
Im zweiten Teil werden die gelernten, gespeicherten Modelle durch den Model-Loader als auch die Lerndaten durch den DBExampleSourceReader geladen. Mit dem ModelApplier werden die Modelle auf den Daten angewendet, und die Vorhersagen werden mit dem ExampleSetWriter gespeichert. Dieser Vorgang wird in diesem Teil für jedes der drei Modelle durchgeführt.
Schließlich wird im dritten Teil über den drei Vorhersagen gelernt (Metalearning). Hierzu werden zunächst die im vorigen Teil gespeicherten Vorhersagen sowie das Zielattribut der Lerndaten mit Hilfe des ExampleSourceReaders geladen. Auf diesem ExmapleSource wird gekapselt durch eine zehnfache Kreuzvalidierung wiederum mit einem J48 Entscheidungsbaumlerner (C 0,95 und M 2) gelernt. Dieses Modell wird als MetaLearningModel gespeichert.

16.4.6 Geschätzte Accuracy des gelernten Modells

Bei der Schätzung der Accuracy des Gesamtmodells treten folgende Schwierigkeiten auf. Es ist zwar möglich, die Accuracy Werte der Kreuzvalidierungen zu übernehmen, allerdings bei den Vorhersagen der Oneclick Sessions mit Bots bzw. der Multiclick Sessions mit Partial Test Patterns kann man nur Aussagen über die Accuracy der vorliegenden Daten machen. Es existiert somit keine "unabhängige" Testmenge, wie sie bei der Kreuzvalidierung besteht. Trotz dieser Problematik wird eine Abschätzung der Gesamtaccuracy durch die Summe

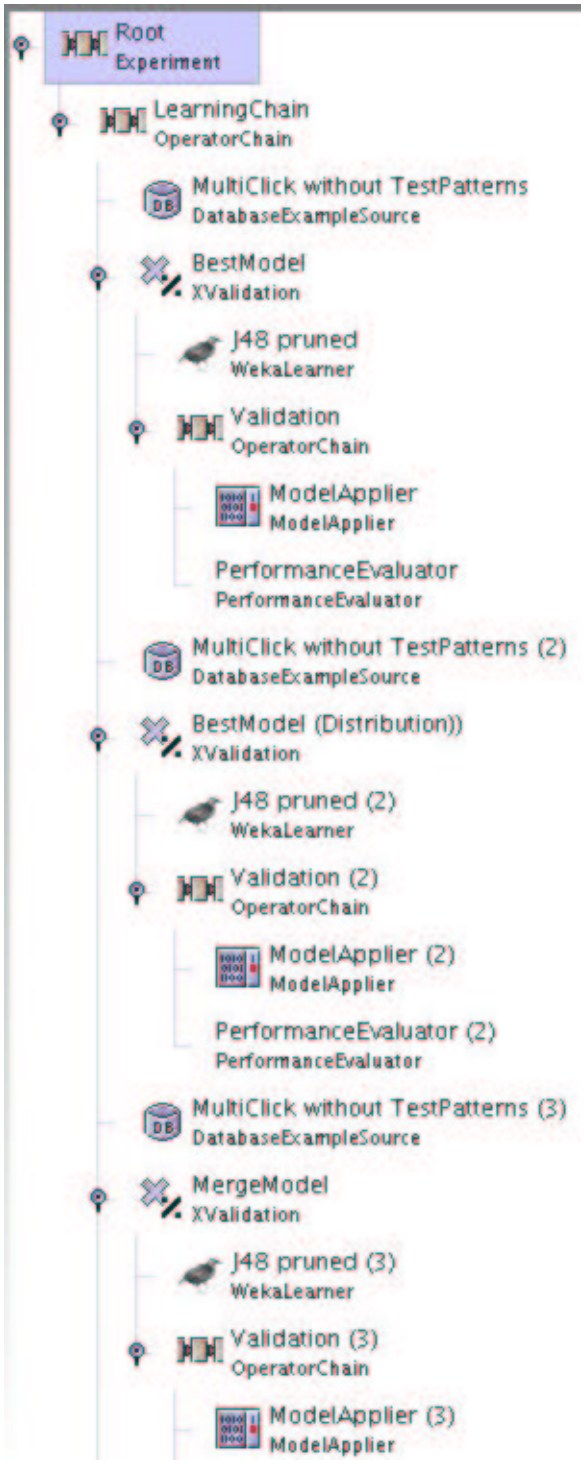


Abbildung 16.4: Yale Operatorkette zum Lernen der Multiclick Sessions Teil 1

über die Produkte aus relativer Häufigkeit und den erhaltenen Accuracies be-

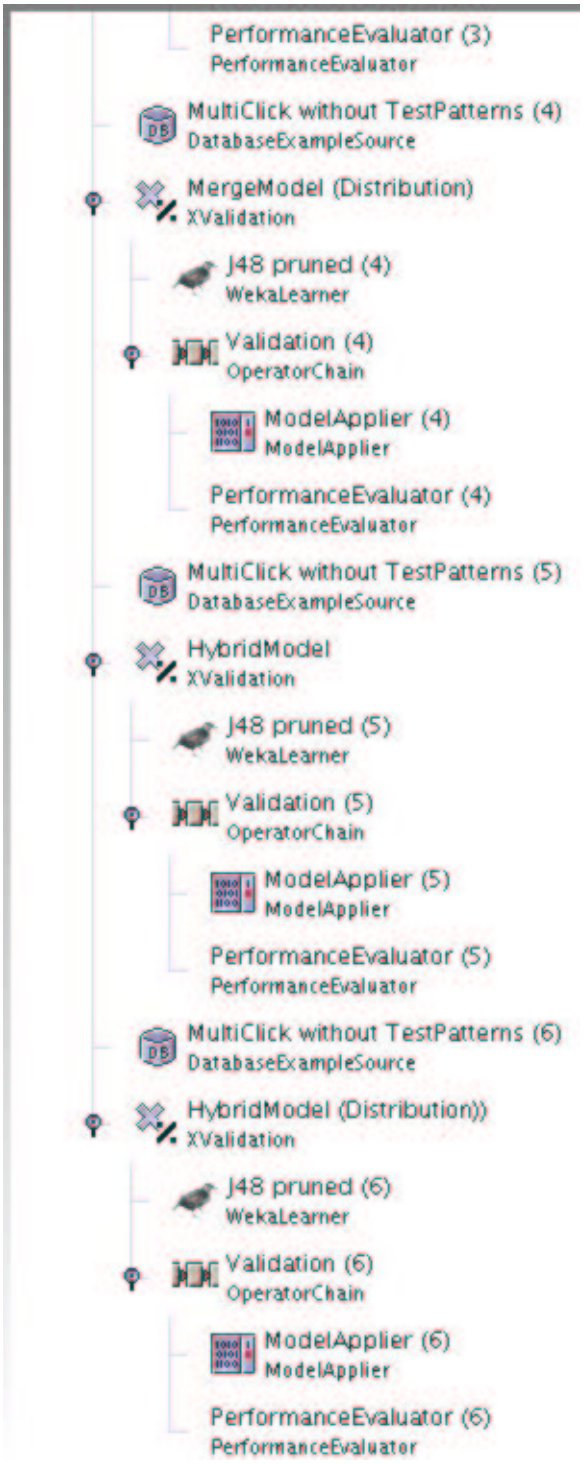


Abbildung 16.5: Yale Operatorkette zum Lernen der Multiclick Sessions Teil 2

rechnet.

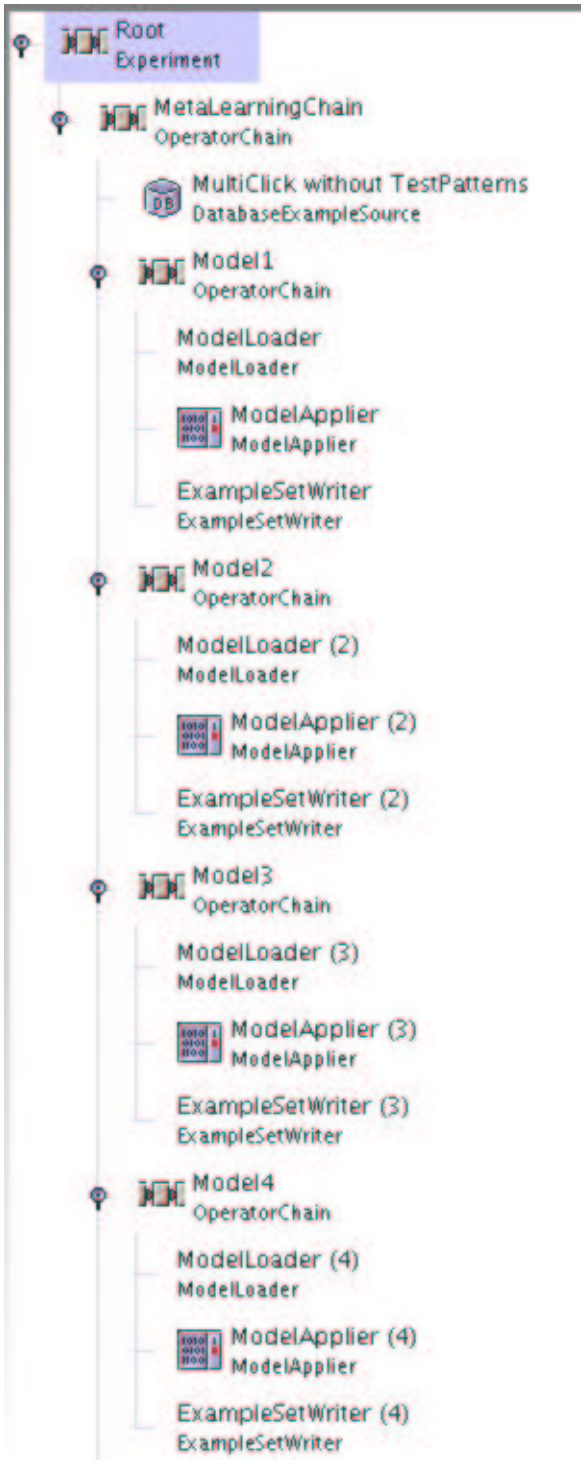


Abbildung 16.6: Yale Operatorkette zum Lernen der Multiclick Sessions Teil 3

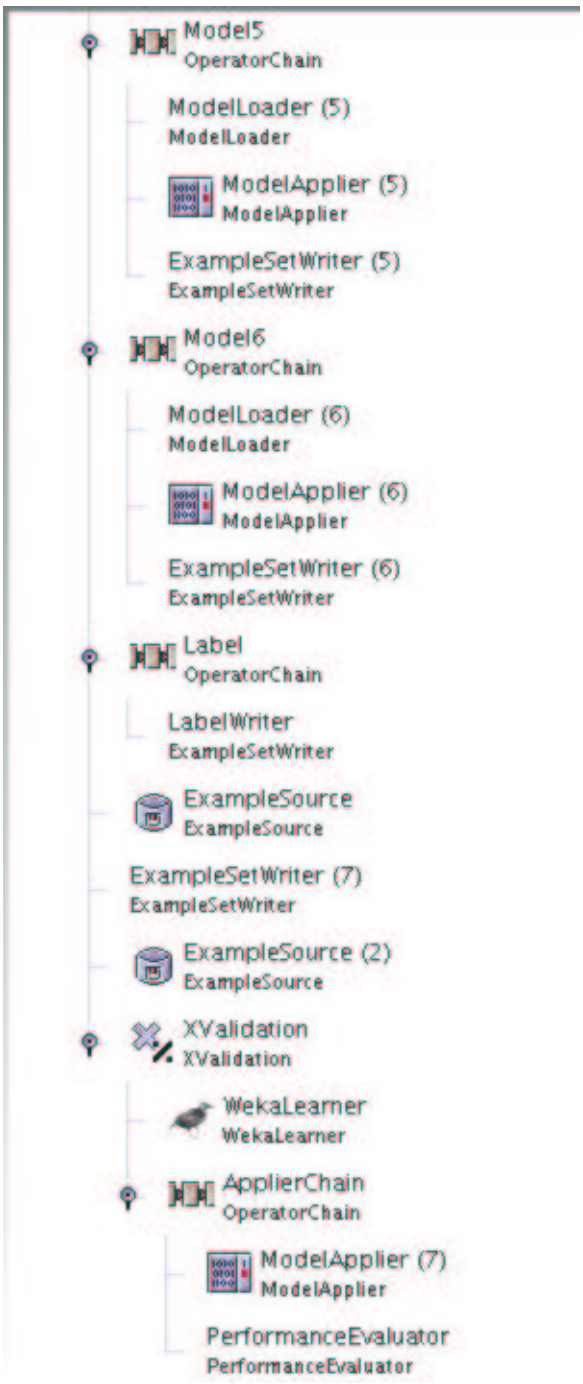


Abbildung 16.7: Yale Operatorkette zum Lernen der Multiclick Sessions Teil 4

Daten	Accuracy	rel. Häufigkeit
Oneclick Session Bots	99,95%	51,26%
Oneclick Sessions ohne Bots	81%	19%
Multiclick Session Test Patterns	42,6%	0,93%
Multiclick Sessions ohne Test Patterns	67,3%	28,81%
Gesamt	86,16%	100%

16.5 Analysekette zur Vorhersage der Testdaten

Die Analysekette zur Vorhersage der Testdaten ähnelt im Aufbau der Analysekette zum Lernen. Die mit MiningMart durchgeführte Trennung der Daten ist identisch. Bei der Operatorkette in Yale zum Lernen der Oneclick Sessions ohne Bots werden die durch die Kreuzvalidierung gekapselten Lernvorgänge durch die Operatoren ModelLoader, ModelApplier und ExampleSetWriter ersetzt. Der ExampleSetWriter speichert die Vorhersage, die durch das geladene Modell erstellt wurde. Bei der Operatorkette zum Lernen der Multiclick Sessions ohne Partial Test Patterns fällt der erste Teil weg. Der zweite Teil bleibt bestehen, hier werden die drei Vorhersagen nach den drei Modellen gebildet und abgespeichert. Diese drei Vorhersagen bilden im dritten Teil zusammen mit dem geladenen MetaLearningModel die Eingabe für einen ModelApplier, der die endgültige Vorhersage trifft und sie speichert.

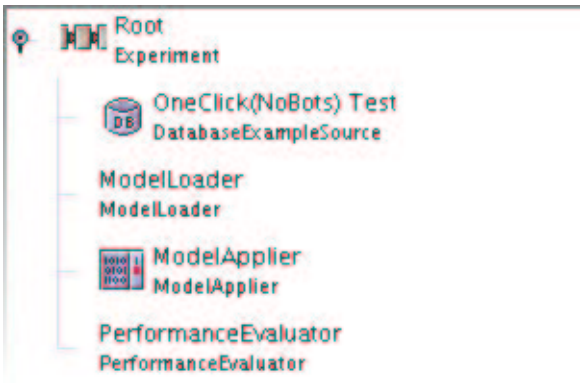


Abbildung 16.8: Yale Operatorkette zum Lernen der Oneclick Sessions

16.5.1 Erreichte Accuracy

Folgende Accuracies wurden mit den beschriebenen Modellen erreicht:

Daten	Accuracy	rel. Häufigkeit
Oneclick Session Bots	99,97%	11,49%
Oneclick Sessions ohne Bots	71,63%	53,43%
Multiclick Session Test Patterns	50%	1,73%
Multiclick Sessions ohne Test Patterns	72,96%	33,36%
Gesamt	74,94%	100%

Beim KDD Cup 2000 entspräche dies dem 16. Platz.

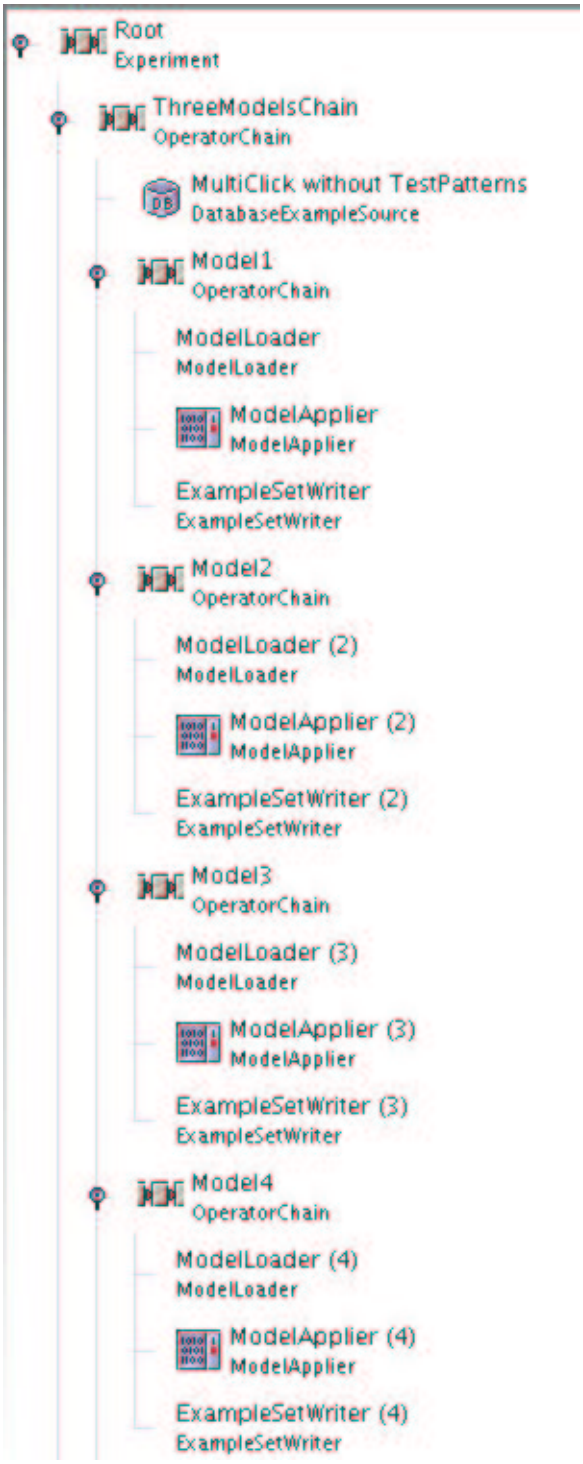


Abbildung 16.9: Yale Operatorkette zum Vorhersage der Multiclick Sessions Teil 1

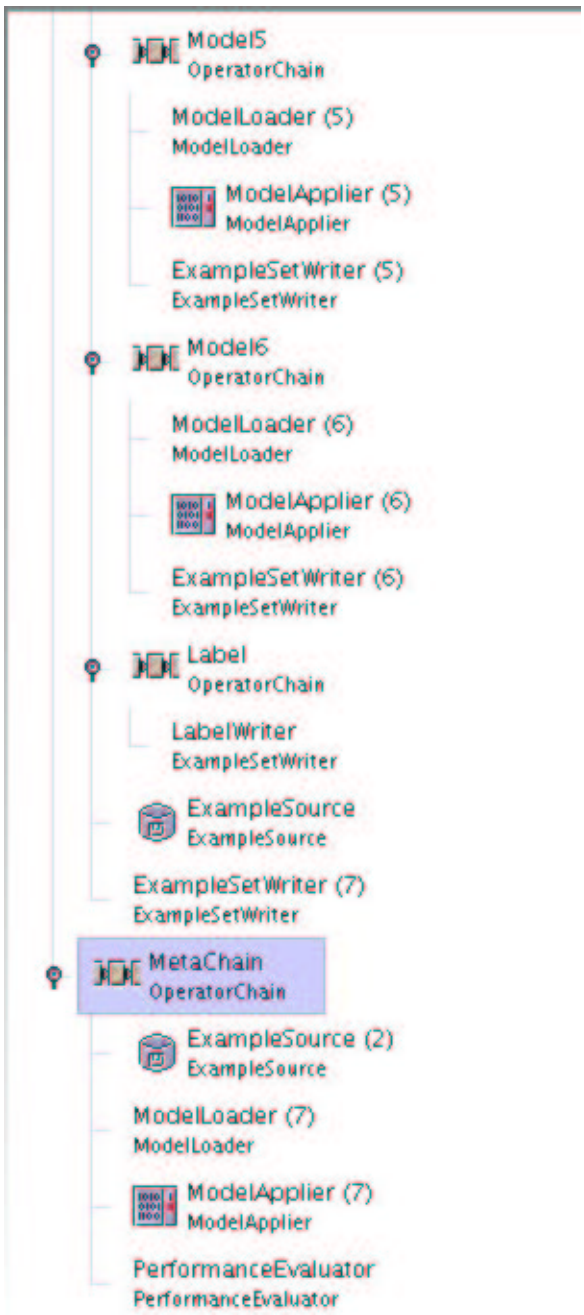


Abbildung 16.10: Yale Operatorkette zum Vorhersage der Multiclick Sessions Teil 2

16.6 Anhang

16.6.1 Weggelassene Attribute bei den Oneclick Sessions

WhichDoYouWearMostFrequent
YourFavoriteLegcareBrand

RegistrationGender
NumberOfChildren
DoYouPurchaseForOthers
HowDoYouDressForWork
HowManyPairsDoYouPurchase
YourFavoriteLegwearBrand
WhoMakesPurchasesForYou
NumberOfAdults2
HowDidYouHearAboutUs
Company
SendEmail
HowOftenDoYouPurchase
HowDidYouFindUs
City
Country
USState
AccountCreationDate
AccountCreationDate_Time
YearofBirth
Email
LoginFailureCount
CustomerID
TruckOwner
RVOwner
MotorcycleOwner
ValueOfAllVehicles
Age
OtherIndiv_Age
MaritalStatus
WorkingWoman
MailResponder
BankCardHolder
GasCardHolder
UpscaleCardHolder
UnknownCardType
TECardHolder
PremiumCardHolder
PresenceOfChildren
NumberOfAdults1
EstimatedIncomeCode
HomeMarketValue
NewCarBuyer
VehicleLifestyle

PropertyType
LoanToValuePercent
PresenceOfPool
YearHouseWasBuilt
OwnOrRentHome
LengthOfResidence
MailOrderBuyer
YearHomeWasBought
HomePurchaseDate
NumberOfVehicles
DMANoMailSolicitationFlag
DMANoPhoneSolicitationFlag
CRAIncomeClassification
NewBankCard
NumberOfCreditLines
SpecialityStoreRetail
OilRetailActivity
BankRetailActivity
FinanceRetailActivity
MiscellaneousRetailActivity
UpscaleRetail
UpscaleSpecialityRetail
RetailActivity
LastRetailDate
LastRetailDate_Time
DwellingSize
DataquickMarketCode
LendableHomeEquity
HomeSizeRange
LotSizeRange
InsuranceExpiryMonth
DwellingUnitSize
MonthHomeWasBought
HouseholdStatus
AvailableHomeEquity
MinorityCensusTract
VerificationDate
VerificationDate_Time
YearOfStructure
Gender
Occupation
OtherIndiv_Gender
OtherIndiv_Occupation

CookieFirstVisitDate
CookieFirstVisitDate_Time
SessionFirstRequestDate
SessionFirstRequestDate_Time
SessionCookieID
SessionID
SessionCustomerID
SessionUserAgent
SessionFirstQueryString
SessionFirstContentID
SessionLastRequestDate
SessionLastRequestDate_Time
SessionLastQueryString
SessionLastRequestDayOfWeek

16.6.2 Weggelassene Attribute bei den Multiclick Sessions

Company
City
AccountCreationDate
AccountCreationDate_Time
CustomerID
VerificationDate
VerificationDate_Time
CookieFirstVisitDate
CookieFirstVisitDate_Time
SessionFirstRequestDate
SessionFirstRequestDate_Time
SessionCookieID
SessionID
SessionCustomerID
SessionUserAgent
SessionFirstQueryString
SessionFirstReferrer
SessionLastRequestDate
SessionLastRequestDate_Time
RequestProcessingTimeAverage
SessionLastQueryString

SessionBrowserFamily
SessionBrowser
SessionBrowserFamilyTop3

Teil III

KDD Cup 2004

Kapitel 17

Quantenphysik-Aufgabe

Marcel Gaspar, Felix Jungermann, Anna Litvina, Lars Michele, Marc Twiehaus, Nazif Veliu

17.1 Die Aufgabe

Die beiden Aufgaben des KDD Cup 2004 zielten auf die Behandlung unterschiedlicher Bewertungsmaße beim überwachten Lernen ab.[1]
Die erste Aufgabe des Cups bestand darin, Klassifizierer für die Unterscheidung von Teilchen eines Hochenergie-Teilchenbeschleuniger Experimentes zu erstellen, unter Bewertung der Vorhersagen mittels folgender Performanzmaße:

- Accuracy (maximieren)
- Fläche unter der ROC-Kurve (AUC) (maximieren)
- Kreuzentropie (minimieren)
- SLAC Q-Score (maximieren)

Der Trainingsdatensatz setzte sich zusammen aus 50000 Beispielen, jedes davon bestehend aus 78 nicht genauer beschriebenen Attributen, einem Label, und

[1] siehe http://kodiak.cs.cornell.edu/kddcup/tasks.html

einer ID als aufsteigende Nummer von 1 bis 50000. Dasselbe galt für den Testdatensatz, nur enthielt dieser 100000 Beispiele mit ID's von 50001 bis 150000 und ohne Label.
Die Abgaben für die 4 Performanzmaße sollten jeweils die Vorhersagen auf dem Testdatensatz sein, jede bestehend aus einer zweispaltigen Datei mit der ID als erster und dem dazugehörigen vohergesagten Label als zweiter Spalte. Alle 4 Performanzmaße konnten (bzw. sollten) getrennt voneinander bearbeitet und eingeschickt werden.
Die Bewertung der Abgaben erfolgte durch das Programm **PERF** [2], welches zum Download zur Verfügung stand.
Es gewann der Teilnehmer/die Teilnehmerin, welche(r) für alle 4 Kriterien die durschnittlich besten Vorhersagen lieferte.
Da unsere Projektgruppe zwei Tage vor dem offiziellen Abgabeschluss einen Fehler in der SLAC Q-Score Berechnung des Programms **PERF** feststellte und dem Veranstalter mitteilte, wurde dieses Maß jedoch bei der Siegerermittlung nicht berücksichtigt.

17.2 Die Bewertungsmaße

Die vier Bewertungskriterien sollen hier nun kurz erläutert werden. Für eine ausführlichere Definition wird auf die Seite des Veranstalters verwiesen.[3]

17.2.1 Accuracy (ACC)

Die Accuracy berechnet sich nach folgender Formel:

$$Accuracy = \frac{\text{Anzahl korrekt vorhergesagter Beispiele}}{\text{Anzahl aller Beispiele}}$$

Die Vorhersage für dieses Kriterium konnte entweder binär (0 oder 1), oder die Wahrscheinlichkeit für die Zugehörigkeit des Beispieles zur Klasse 1 sein.
Bei Verwendung der Wahrscheinlichkeitswerte gehört zur Umrechnung in eine binäre Vorhersage ein Schwellwert, ab welchem ein Beispiel als zur Klasse 1 zugehörig betrachtet wird.

[2] siehe http://kodiak.cs.cornell.edu/kddcup/software.html
[3] siehe http://kodiak.cs.cornell.edu/kddcup/

17.2.2 Fläche unter ROC-Kurve (AUC)

Die Güte eines "scharfen" Klassifizierers (gibt die Klasse eines Beispieles direkt an) kann angegeben werden durch ein Tupel der beiden Werte True-Positive-Rate (TPR) und False-Positive-Rate (FPR), die sich nach folgenden Formeln berechnen lassen:

$$TPR = \frac{\text{Anzahl der als pos. klassifizierten Beispiele, die tatsächlich pos. sind}}{\text{Anzahl aller tatsächlich positiven Beispiele}}$$

$$FPR = \frac{\text{Anzahl der als pos. klassifizierten Beispiele, die tatsächlich neg. sind}}{\text{Anzahl aller tatsächlich negativen Beispiele}}$$

In einem Koordinatensystem, in dem FPR auf der Abszisse, und TPR auf der Ordinate abgetragen wird, entspricht dieses Tupel einem Punkt.
Ein "weicher" Klassifizierer (gibt die Wahrscheinlichkeit der Zugehörigkeit zu einer Klasse an) läßt sich nach Angabe eines Schwellwertes (siehe Accuracy) auf die gleiche Weise durch einen Punkt im TPR-FPR-Koordinatensystem charakterisieren.
Wenn nun der Schwellwert, beginnend bei 0, sukzessiv bis zum Wert 1 erhöht, und für jede Erhöhung das zugehörige TPR-FPR Tupel berechnet und als Punkt abgetragen wird, erhält man eine Möglichkeit, die sogenannte ROC (Receiver Operating Characteristics) -Kurve zu konstruieren.
Die Fläche unter der ROC-Kurve (AUC = Area Under Curve) entspricht der Fläche unter dieser Kurve.
Da TPR und FPR normalisierte Werte sind, liegt AUC im Intervall [0,1], wobei eine zufällige Klassifikation ohne Bezug zur wahren Klasse in einem Wert von 0,5 resultiert.

17.2.3 Kreuzentropie (CXE)

Die Kreuzentropie mißt, wie nahe die vorhergesagten Werte an den tatsächlichen Werten liegen. Im Unterschied zum quadratischen Fehler wird bei der Kreuzentropie davon ausgegangen, dass die Vorhersage im Intervall [0,1] liegt, und die Wahrscheinlichkeit der Zugehörigkeit des jeweiligen Falles zur Klasse 1 angibt. Der Wert berechnet sich nach folgender Formel:

$$Kreuzentropie = \sum ((x) * log(y) + (1 - x) * log(1 - y))$$

wobei x die tatsächliche Klasse (0 oder 1) und y die Vorhersage für jeweils ein Beispiel angibt.

Um einen von der Größe des Datensatzes unabhängigen Wert für die Kreuzentropie zu erhalten, wird in diesem Cup die mittlere Kreuzentropie benutzt, welche der obigen Summe geteilt durch die Anzahl der Beispiele des Datensatzes entspricht.

17.2.4 Slac Q-Score (SLQ)

Slac Q-Score ist ein anwendungsspezifisches Bewertungsmaß, welches von Forschern des Stanford Linear Accellerator (SLAC) entwickelt wurde, um die Leistung von Vorhersagen für diverse Probleme der Teilchenphysik zu messen. Wie die Kreuzentropie arbeitet dieses Bewertungsmaß auf Vorhersagen im Intervall [0,1].
Aufgrund der ansonsten eher geringen Relevanz dieses Kriteriums für den Bereich KDD gehen wir an dieser Stelle nicht näher darauf ein, sondern verweisen für eine genaue Definition auf die Homepage des Veranstalters[4].

17.3 Dateninspektion

Es handelt sich um ein binäres Klassifikationsproblem, wobei die 78 Attribute des Trainings- und Testdatensatzes reellwertig sind.
Für den Trainingsdatensatz wurde folgende statistische Analyse durchgeführt:

- für jedes Attribut drei Histograme: eines ohne Berücksichtigung des Labels, eines unter ausschließlicher Betrachtung der Beispiele mit Label 0, und eines unter ausschließlicher Betrachtung der Beispiele mit Label 1
- für jedes Attribut ein Streudiagramm, wobei der Attributwert auf der Abszisse und das Label auf der Ordinate abgetragen ist
- Zusammenfassung des Datensatzes mit Minimum, 1.Quantil, Median, Mittelwert, 3.Quantil und Maximum für jedes Attribut

Die Auswertung ergab folgendes:

- 24861 der insgesamt 50000 Beispiele sind positiv, der Rest negativ

[4] siehe http://kodiak.cs.cornell.edu/kddcup/metrics.html

- die Formen der Histogramme unter ausschließlicher Betrachtung der positiven Beispiele sind den Formen der Histogramme unter ausschließlicher Betrachtung der negativen Beispiele sehr ähnlich
- es existieren keine offensichtlichen Zusammenhänge zwischen einzelnen Attributen und dem Label
- 2 Attribute besitzen als Werte nur 0 und 9999 (Attribute 29 und 55)
- in 6 Attributen tritt in ca 2/3 der Beispiele des Datensatzes der Wert 999 auf (Attribute 20, 21, 22, 44, 45, 46). Zu bemerken ist auch, dass die Attributtripel 20, 21, 22 und 44, 45, 46 jeweils immer gleichzeitig mit dem Wert 999 belegt sind.
- 5 Attribute besitzen als einzige Ausprägung den Wert 0 (Attribute 47, 48, 49, 50, 51)

Für den Testdatensatz wurden folgende Analysen durchgeführt:

- für jedes Attribut ein Histogramm
- Zusammenfassung des Datensatzes mit Minimum, 1.Quantil, Median, Mittelwert, 3.Quantil und Maximum für jedes der Attribute

Die Auswertung ergab folgendes:

- die Formen der Histogramme des Testdatensatzes sind für alle Attribute den Formen der Histogramme des Trainingsdatensatzes sehr ähnlich
- 2 Attribute besitzen als Werte nur 0 und 9999 (Attribute 29 und 55)
- in 6 Attributen tritt in ca 2/3 der Beispiele des Datensatzes der Wert 999 auf (Attribute 20, 21, 22, 44, 45, 46). Zu bemerken ist auch, dass die Attributtripel 20, 21, 22 und 44, 45, 46 jeweils immer gleichzeitig mit dem Wert 999 belegt sind.
- 5 Attribute besitzen als einzige Ausprägung den Wert 0 (Attribute 47, 48, 49, 50, 51)

Im nachhinein wurde bekanntgegeben, dass der Datensatz auch fehlende Werte enthält, welche den oben aufgeführten Werten 999 für die Attribute 20, 21, 22, 44, 45, 46 und 9999 für die Attribute 29, 55 entsprechen.

17.4 Vorverarbeitung

Aufgrund der im letzten Abschnitt beschriebenen Beobachtungen führten wir folgende Vorverarbeitungsschritte durch:

1. Die Attribute 29, 47, 48, 49, 50, 51, 55 wurden entfernt, da in 5 Fällen ihre Werte immer 0 waren, und in 2 Fällen ihre Werte entweder 0 waren oder fehlten.

2. In den Attributen 20, 21, 22, 44, 45, 46 des Trainingsdatensatzes wurden die fehlenden Werte durch lineare Regression gelernt.
   Ein im Anschluss daran durchgeführter kreuzvalidierter Lernlauf, einmal mit den nachgelernten Werten und einmal ohne diese, zeigte folgende Verbesserung für den Fall der gelernten Attributwerte:
   Verbesserung für Accuracy: 18%-19%, Fläche unter ROC-Kurve: 15% - 17%, SLQ-Score: 36% - 37%.
   Aufgrund der offensichtlichen Verbesserung wurde im Folgenden ausschließlich dieser modifizierte Datensatz weiterverwendet.

3. Entfernen der Attribute bzw. Lernen der fehlenden Werte derselben Attribute wie in 1. und 2. für den Testdatensatz

Das Erlernen der fehlenden Werte für Trainings- und Testdatensatz erfolgte dabei folgendermaßen (exemplarisch dargestellt für Attribut 20):
Auf der Teilmenge aller Beispiele, für die Attribut 20 einen Wert besaß (ca. 1/3 des jeweiligen gesamten Datensatzes), erstellten wir mittels linearer Regression ein Modell mit Attribut 20 als Zielvariable. Zum Lernen des Modells entfernten wir dabei die Attribute 21, 22, 44, 45 und 46, da sie in den Beispielen, auf die das Modell später angewendet werden sollte, selbst fehlende Werte aufwiesen (siehe Auswertung in der Dateninspektion).
Dieses Modell wurde anschließend auf die restlichen Beispiele des Datensatzes angewendet, um Attribut 20 jeweils mit einem Wert zu belegen.
Dieselbe Prozedur, angewendet auf die restlichen Attribute mit fehlenden Werten, lieferte den entsprechend modifizierten Trainings- und Testdatensatz.

17.5 Durchgeführte Ansätze

Wir überlegten, die Bewertungsmaße vorerst folgendermaßen separat zu bearbeiten:

- Verschiedene Lernalgorithmen wurden einzeln auf ihre Effizienz getestet. Ein J48-Entscheidungsbaum alleine lieferte schon gute Ergebnisse für Accuracy und AUC (siehe Tabelle 17.4)
  Das bezüglich Accuracy und AUC beste Ergebnis lieferte AdaBoost mit J48.
  Für Kreuzentropie und SLQ-Score lieferte kein Verfahren gute Ergebnisse.

- Mit dem Tool *smiles*[5], einem Entscheidungsbaum-Lernsystem mit (u.a.) variabel einstellbarem Aufteilungskriterium bei der Knotenerzeugung, wurde versucht, AUC zu optimieren (lokale AUC-Optimierung bei der Knotenerzeugung).
  Trotz der anfangs leicht verbesserten Ergebnisse wurden die Versuche mit diesem Programm allerdings aufgrund der fehlenden Möglichkeit der Weiterverwendung der erstellten Modelle und dem fehlerhaften Umgang mit fehlenden Werten eingestellt.

- Weiterhin versuchten wir, AUC dadurch zu optimieren, dass auf Stichproben mit einer manuell erstellten unterschiedlichen Verteilung der Klassen gelernt wurde, und diese Vorhersagen dann zu kombinieren (konvexe Hülle der einzelnen ROC-Kurven).
  Die Idee, die einzelnen Stichproben vom Lerner automatisch erstellen zu lassen, führte uns dann jedoch zur Einbeziehung des bis dahin neuen Yale-Operators Bayesian-Boosting, welcher für AUC bessere Ergebnisse lieferte als AdaBoost.

- Ein angedachtes Clustering in Weka lieferte keine guten Ergebnisse.

- Die Überlegung, mehrere Lernverfahren zu kombinieren, führte uns schliesslich zu dem im folgenden Abschnitt dargestellten Ensemble-Ansatz mit (evolutionärer) Gewichtung von Einzelvorhersagen.
  Anfangs lieferte diese Lösung vielversprechende Ergebnisse für SLQ-Score, musste aber zwei Tage vor Abgabeschluss leider verworfen werden, da es sich bei den guten Ergebnissen um einen Fehler im Bewertungsprogramm **PERF** handelte.
  Für die anderen Bewertungsmaße lieferte das Verfahren entgegen allen Erwartungen teilweise schlechtere Ergebnisse als die verwendeten Einzelvorhersagen, worin wir einen möglichen Fehler des Yale-Operators "Evolutionary Weighting" vermuteten.
  Für die Lösung wurde schließlich der im Abschnitt 17.6 auf Seite 294 beschriebene Bagging-Ansatz gewählt.

[5] siehe http://www.dsic.upv.es/ flip/smiles/

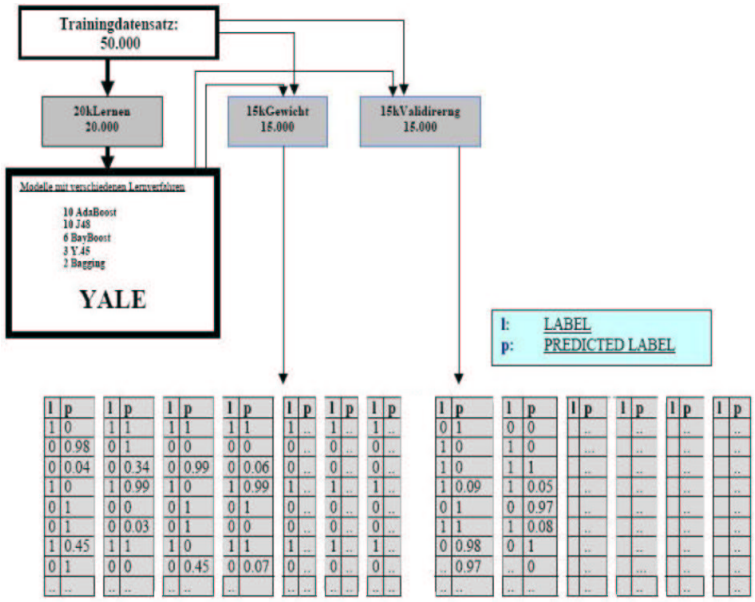


Abbildung 17.1: Schema des Ensemble-Ansatzes

17.5.1 Der Ensemble-Ansatz

Nachdem wir mit einzelnen Modellen zwar schon recht gute Ergebnisse erhielten, haben wir uns dafür entschieden, verschiedene Modelle durch ein Ensemble zu verbinden. Bestärkt wurden wir bei dieser Vorgehensweise durch den Artikel "Ensemble Selection from Libraries of Models" von Rich Caruana etal[12]. Da wir allerdings nicht die Möglichkeiten hatten, die wie im Artikel vorgeschlagenen 2000 Modelle zu trainieren, beschränkten wir uns darauf, nur Bäume in die Ensemble-Methode aufzunehmen, da diese alleine schon gute Ergebnisse lieferten.

Sampling

Aus dem vorverarbeiteten Datensatz, der 50.000 Beispiele enthielt, wurden im ersten Schritt drei disjunkte Stichproben gezogen. Die erste Stichprobe (20kLernen.dat) enthielt 20.000 Datensätze, die zweite (15kGewicht.dat) und dritte (15kValidierung.dat) enthielten je 15.000 Datensätze (siehe Abbildung 17.1). Beim Sampling wurde darauf geachtet, dass die Beispiele (positive und negative) in den einzelnen Samples gleichverteilt bleiben, so wie dies auch im ursprünglichen Trainingsdatensatz der Fall war.

AdaBoost	**J48**	**Bagging**
Parametereinstellungen	***Parametereinstellungen***	***Basislerner***
P=100, I=10, C=0.1, M=3	C=0.25, M=2	J48
P=100, I=9, C=0.2, M=3	C=0.25, M=5	Decision Stumps
P=100, I=8, C=0.3, M=3	C=0.25, M=10	
P=100, I=7, C=0.3, M=5	C=0.25, M=20	
P=100, I=6, C=0.3, M=10	C=0.5, M=2	
P=90, I=10, C=0.1, M=35	C=0.5, M=5	
P=80, I=10, C=0.2, M=3	C=0.5, M=10	
P=70, I=10, C=0.3, M=3	C=0.75, M=2	
P=60, I=10, C=0.3, M=5	C=0.75, M=5	
P=50, I=10, C=0.3, M=10	C=0.75, M=10	

Tabelle 17.1: Parametereinstellung für AdaBoost, J48 und Bagging

Bayesian-Boost	**Y45**
Basislerner	***Parametereinstellungen***
J48	C=0.25, M=2
PART-Regellerner	C=0.5, M=3
Logistic-Funktion	C=0.5, M=2
Random-Forest	
CostSensitiveClassifier und J48, Gewichtung 5:1	
CostSensitiveClassifier und J48, Gewichtung 1:5	

Tabelle 17.2: Parametereinstellung für Bayesian-Boosting und Y45

Lernschritt

In einer ersten YALE-Kette (siehe Abbildung 17.2) wurden 31 Modelle mit den fünf verschiedenen Lernmethoden AdaBoost, J48, Bayesian-Boosting, Y45 und Bagging erstellt. Bei diesen Lernmethoden wurden verschiedene Einstellungen (siehe Tabellen 17.1 und 17.2) eingesetzt.

Beispielsweise wurden für die Lernmethode AdaBoost, die als Basislerner J48 hatte, die vier Parametereinstellungen I (Anzahl der Iterationen), C (Konfidenz für das Beschneiden der Bäume), M (Anzahl der Beispiele pro Blatt) und P variiert. Dabei gehörten die Parameter P und I zu AdaBoost und die Parameter C und M zum J48-Basislerner.

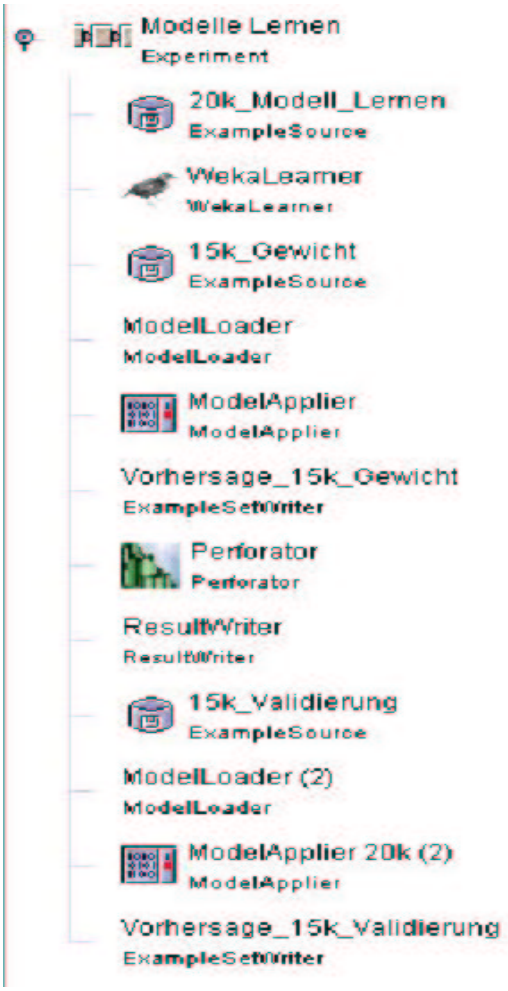


Abbildung 17.2: Yale-Kette, die zum Lernen und Anwenden der Modelle benutzt wurde

Anwendungsschritt

Nach dem Lernen wurden alle Modelle auf die beiden Datensätze (15kGewicht.dat und 15kValidierung.dat) angewandt (siehe Abbildung 17.2). Jedes Modell lieferte eine Vorhersage zu den zwei unabhängigen Datensätzen.

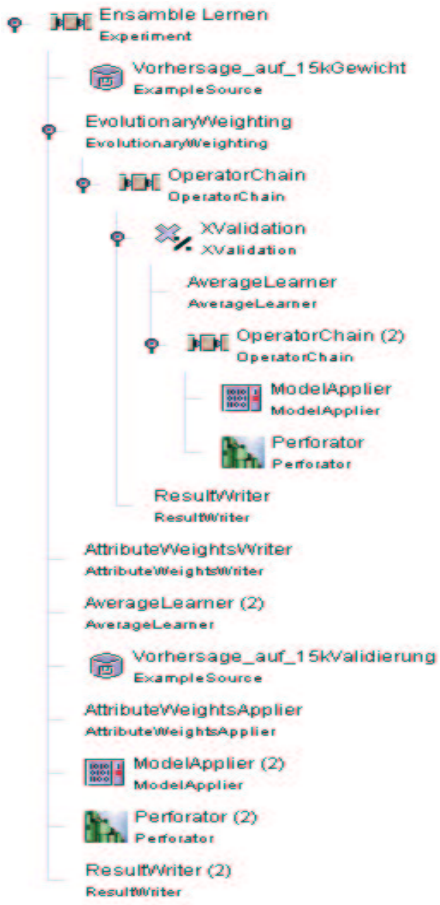


Abbildung 17.3: Yale-Kette, die zur Optimierung der Bewertungsmaße diente

Optimierungsschritt

In einer zweiten YALE-Kette (siehe Abbildung 17.3) wurden dann die im Anwendungsschritt erstellten Vorhersagen zusammengefasst. Anschließend wurde mit Hilfe des AverageLearners der beste gewichtete Durchschnitt für den ersten Datensatz (15kGewicht.dat) in Bezug auf ein Performancemaß erstellt.

Dadurch erhält man für die einzelnen Bewertungskriterien ACC (Accuracy), AUC (Fläche unter ROC-Kurve), CXE (Kreuzentropie) und SLQ einen Gewichtsvektor, welcher auf den Validierungsdatensatz (15kValidierung.dat) angewandt wird und auf diese Weise die endgültigen Ergebnisse für das trainierte bzw. optimierte Bewertungsmaß liefert.

Performance-Maße	Ergebnisse
ACC	0.83 - 0.87
ROC	0.91 - 0.93
CXE	0.45 - 0.48
SLQ	0.65 - 0.68

Tabelle 17.3: Ergebnisse mit zehnfacher Kreuzvalidierung

Ergebnisse

Das dargestellte Verfahren wurde mittels 10-facher Kreuzvalidierung überprüft, was zu den in der Tabelle dargestellten Ergebniswerten führte. Leider erreichte diese Methode aber schlechtere Ergebnisse als das endgültig gewählte und im folgenden Abschnitt beschriebene Verfahren.

17.6 Der Lösungsansatz

Nachdem zuvor versucht wurde unterschiedliche Modelle auf einem Sample zu lernen und diese dann zu einem Ensemble zu verbinden, entschied sich die Gruppe für einen anderen Ansatz. Der Ensemble Ansatz erschien grundsätzlich weiterhin sinnvoll, er lieferte bessere Ergebnisse als jedes einzelne Modell und, im Gegensatz zum besten Einzel-Modell (Bayboost mit J48), nicht nur binäre Vorhersagen (die schlecht für slq und cxe waren). Daher war der logische Schluß nicht die Abstimmung abzuschaffen, sondern deren Teilnehmer auszutauschen. Die Vermutung war nun, daß es vielleicht besser wäre statt nur ein Modell auf einem Sample zu lernen, einen Lernalgorithmus auf mehreren Samples anzusetzen und dann so das Komitee mit verschiedenen Gruppen von Modellen eines Typs zu füllen. Dazu wurde zunächst der vorverarbeitete Trainingsdatensatz in zwei Teile der Größe 10.000 und 40.000 geteilt, die 10.000 Instanzen bildeten das Validierungs-Sample und wurden nur zur endgültigen Validierung der Ergebnisse verwendet. Die Instanzen aus dem 40.000er Sample (im folgenden als Gewichts-Sample bezeichnet) wurden dagegen sowohl zum Lernen der Modelle als auch zum Lernen des Gewichtsvektors benutzt und dienten als neues Trainingsset. Hierzu wurden insgesamt fünf Samples der Größe 30.000 (im folgenden mit 30k1 bis 30k5 bezeichnet) aus dem Gewichts-Sample gezogen, jeweils unabhängig und annähernd gleichverteilt. Auf diesen fünf Samples wurden dann jeweils sechs Modell-Gruppen gelernt, so dass also

insgesamt 30 Modelle gelernt wurden, die dann das Ausgangs-Ensemble bildeten. Die sechs Gruppen im einzelnen:

- AdaboostM1 mit J48 als Basislerner, für die Anzahl der Adaboost Iterationen wurde 10 gewählt, die Parameter für den J48 Lerner waren confidence 0.25 und min-leaf-size 2
- Bayboost mit J48 als Basislerner, mit 10 Iterationen und J48 mit confidence 0.25 und min-leaf-size 2
- J48 mit confidence 0.25 und min-leaf-size 2
- Y45 mit confidence 0.25, min-leaf-size 5, reduced folds 3 und reduced error pruning
- AdaboostM1 mit decision stumps als Basislerner, wiederum 10 Iterationen für Adaboost
- Adtree (alternating decision tree), ebenfalls 10 Iterationen.

Weiterhin wurden die Modelle so verwendet, daß jeweils AdaboostM1 mit J48 und AdaboostM1 mit decision stumps binäre Vorhersagen machten, die anderen vier Modelle aber die Konfidenz dafür ausgaben, daß die Vorhersage Klasse 1 entspricht (eine Zahl im Intervall zwischen 0 und 1). Hier die gerundeten durchschnittlichen Ergebnisse der einzelnen Modell-Gruppen:

	ACC	ROCA	CXE	SLQ
Adaboost (J48)	~ 0.85	~ 0.85	-	~ 0.65
Adtree	~ 0.75	~ 0.81	-	~ 0.3
Bayboost	~ 0.88	~ 0.93	-	~ 0.85
J48	~ 0.83	~ 0.84	-	~ 0.6
Adaboost (DecisionStumps)	~ 0.69	~ 0.69	-	~ 0.36
y45	~ 0.64	~ 0.63	-	~ 0.11

Tabelle 17.4: durchschnittliche Ergebnisse der Model-Gruppen

Die fehlenden Werte bei cxe resultieren daher, dass die Berechnung des Wertes für ein einzelnens Modell entweder nahe bei 1 lag oder Werte bis zu 9 Milliarden annahm, also entweder sehr schlecht oder sogar absolut unbrauchbar war.

Jeder dieser Modell-Typen wurde fünf mal auf den unterschiedlichen Samples gelernt, so daß wir insgesamt 30 Modelle als Ausgang für unser Ensemble erhielten. Dadurch sollte erreicht werden, dass die einzelnen Modell-Gruppen jeweils einen anderen Teil der Trainingsdaten lernten, die Gruppe aber alle Trainingsdaten abdecken würde.
Das Finden des Gewichtsvektors wurde auf dem 40.000er Sample durchgeführt, so dass die dazu verwendeten Modelle nicht komplett auf unbekannten Daten angewandt wurden, aber, um Überanpassung zu vermeiden, jedes einzelne Modell 10.000 Instanzen der Daten nicht kannte. Die endgültige Vorhersage auf dem Validierungs-Sample wurde dann als Durchschnitt über die gewichteten Vorhersagen der 30 Einzel-Modelle berechnet, wobei die Vorhersage für eine Instanz i der Summe der gewichteten Vorhersagen der Modelle für Instanz i geteilt durch die Anzahl der verwendeten Gewichte entsprach, also:

$$pred(i) = \frac{\sum_{k=1}^{30} PredModel_k(i) * Gewicht_k}{\sharp(Gewichte \neq Null)}$$

17.6.1 Modellierung in Yale

Insgesamt wurden in Yale sechs Ketten für die Modellerzeugung, eine Kette für das Lernen des Gewichtsvektors und eine weitere Kette für die Auswertung auf dem Validierungsset bzw. für die endgültige Vorhersage auf dem Testset erstellt.

Ketten 1.1 bis 1.6: Erstellung der 30 Ausgangsmodelle

Die abgebildete Kette wurde insgesamt sechs mal verwendet, wobei der in der Abbildung enthaltene Lernalgorithmus in den anderen fünf Ketten durch einen der anderen, oben genannten ersetzt wurde.

Eingang:

- Die 5 Samples 30k1 bis 30k5 (⇒ ExampleSource)
- Das 40.000er Sample zum Gewichtlernen (⇒ ExampleSource (2))
- Das 10.000er Validierungs-Sample (⇒ ExampleSource (3))

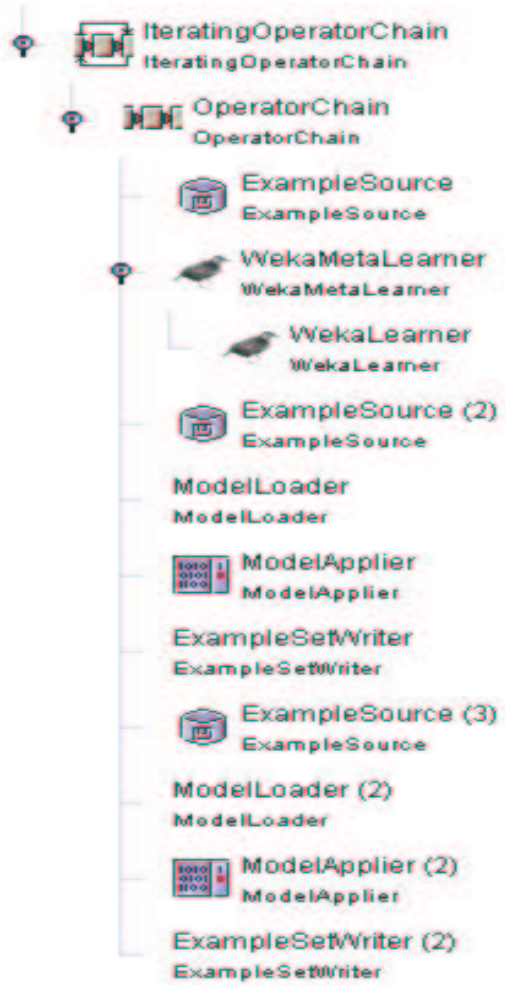


Abbildung 17.4: Lernen von 5 Basismodellen

Ausgang:

- 5 Modelle eines Typs, gelernt auf den Samples 30k1 bis 30k5 (⇒ Learner)
- Die 5 Vorhersagen der Modelle auf dem Gewichts-Sample (⇒ ExampleSetWriter)
- Die 5 Vorhersagen auf dem Validierungs-Sample (⇒ ExampleSetWriter (2))

Die Modelle wurden aus praktischen Gründen auch gleich auf dem Gewichts- und dem Validierungs-Sample angewandt. Die Vorhersagen der Modelle auf

dem Gewichts-Sample wurden in Kette 2, die auf dem Validierungs-Sample in Kette 3 benötigt.

Kette: Lernen des Gewichtsvektors

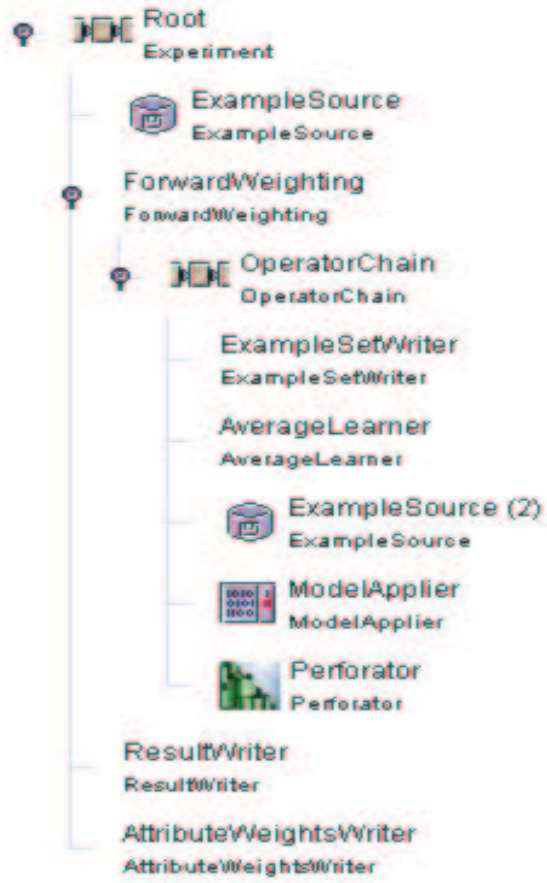


Abbildung 17.5: Lernen des Gewichtsvektors

Die Vorhersage der 30 Modelle auf dem Gewichts-Sample wurden zu einem neuen ExampleSet zusammengefaßt, so dass die Vorhersage eines jeden Modells als Attribut auftrat und das wahre Label aus dem Gewichts-Sample übernommen wurde. Anschließend für jedes der Bewertungskriterien mittels einer Forward-Attribut-Selection ein optimaler Gewichtsvektor für die Vorhersagen gesucht. Die Forward-Selection wurde dadurch eingeschränkt, daß wir die Gewichte fest vorgaben. Diese Gewichte waren:
0 0.1 0.01 0.001 0.2 0.3 0.4 0.45 0.5 0.55 0.6 0.65 0.7 0.75 0.8 0.85 0.9 0.925 0.95 0.97 0.98 0.99 0.999 0.9999 1.0.

Dies sollte zum einen die mögliche Suche nach einem optimalen Vektor zeitlich beschränken, zum anderen sollte dadurch verhindert werden, dass die Forward-Selection negative Gewichte auswählte, die die Vorhersagen für den Perforator unbrauchbar machten.

Eingang:

- Ein Meta-Attribut-Set mit den Vorhersagen der 30 Modelle auf dem Gewichts-Sample und dem Label des Gewichts-Samples (⇒ Example-Source)

Ausgang:

- Der optimierte Gewichtsvektor (⇒ AttributsWeightsWriter)

Kette: Validierung des Gewichtsvektors

Schließlich wurde der Gewichtsvektor auf dem Validierungs-Sample angewandt. Dazu wurden die bereits bei der Modell-Erstellung erstellten Vorhersagen der 30 einzelnen Modelle wiederum zu einem Example-Set zusammengefasst, wobei das Label diesmal aus dem Validierungs-Sample stammt. Dann wurde der Gewichtsvektor mittels eines Attribut-Weight-Appliers angewandt, um anschließend den Durchschnitt der gewichteten Attribute mittels eines AverageLearnes zu berechnen.

Eingang:

- Ein Meta-Attribut-Set mit den Vorhersagen der 30 Modelle auf dem Validierungs-Sample (⇒ ExampleSource)
- Der optimierte Gewichtsvektor (⇒ AttributeWeightsLoader)

Ausgang:

- Die Vorhersage (⇒ ExampleSetWriter), wobei die Vorhersage für Instanz i der Summe der gewichteten Vorhersagen Modelle für Instanz i geteilt durch die Anzahl der verwendeten Gewichte entsprach, also

Abbildung 17.6: Validierung des Gewichtsvektors

$$pred(i) = \frac{\sum_{k=1}^{30} PredModel_k(i) * Gewicht_k}{\sharp(Gewichte \neq Null)}$$

Hier die Ergebnisse auf dem Validierungsset:

	Ensemble	ungewichteter Durchschn.	bestes Einzel-Modell
ACC	0.874	0.86	~ 0.85
CXE	0.426	0.514	~ 0.65
ROCA	0.946	0.927	~ 0.91
SLQ	0.768	0.597	~ 0.614

Dabei wurden die Ergebnisse für den ungewichteten Durchschnitt sowie das

beste Modell ohne die Bayboost Modelle errechnet (Die Bayboost-Modelle brauchten am längsten, diese Zwischenergebnisse wurden während der Wartezeit erstellt).

Das endgültige Ensemble

Eine von uns durchgeführte 5-fache Kreuzvalidierung dieses Ansatzes bestätigte die erhaltenen Ergebnisse und somit wurde beschlossen, diesen Ansatz für die Erstellung der endgültigen Abgabe zu verwenden. Hier die Ergebnisse der Kreuzvalidierung (bis auf die letzte Zeile handelt es sich bei den Werten in den Tabellen um die jeweiligen Trainingsfehler, in der letzten Zeile stehen die Ergebnisse für die Validierungsstichprobe):

Lerner	Parametereinstellungen
Adaboost	J48 Basis Lerner, C=0.25, M=2, I=10
Adtree	I=10
Bayboost	J48 Basis Lerner, C=0.25, M=2, I=10
J48	C=0.25, M=2
Adaboost	DecisionStumps Basis Lerner, I=10
y45	C=0.25, M=5, reduced folds 3, reduces error pruning

1. Wiederholung	ACC	ROCA	CXE	SLQ
Adaboost (J48)	0.963	0.963	...	0.851
Adtree	0.751	0.815	...	0.310
Bayboost	0.966	0.992	...	0.857
J48	0.936	0.956	...	0.759
Adaboost (DecisionStumps)	0.695	0.695	...	0.359
y45	0.643	0.633	...	0.111
Forward Weighting	0.995	1.000	0.121	0.980
Validierung	0.878	0.952	0.416	0.980

2. Wiederholung	ACC	ROCA	CXE	SLQ
Adaboost (J48)	0.963	0.963	...	0.853
Adtree	0.749	0.814	...	0.308
Bayboost	0.967	0.992	...	0.859
J48	0.934	0.953	...	0.752
Adaboost (DecisionStumps)	0.704	0.705	...	0.222
y45	0.588	0.587	...	0.013
Forward Weighting	0.995	1.000	0.114	0.988
Validierung	0.880	0.957	0.408	0.670

3. Wiederholung	ACC	ROCA	CXE	SLQ
Adaboost (J48)	0.963	0.963	...	0.860
Adtree	0.746	0.812	...	0.304
Bayboost	0.964	0.99	...	0.858
J48	0.931	0.952	...	0.746
Adaboost (DecisionStumps)	0.727	0.726	...	0.156
y45	0.641	0.666	...	0.146
Forward Weighting	0.995	1.000	0.114	0.985
Validierung	0.881	0.955	0.405	0.670

4. Wiederholung	ACC	ROCA	CXE	SLQ
Adaboost (J48)	0.962	0.962	...	0.856
Adtree	0.747	0.814	...	0.310
Bayboost	0.966	0.992	...	0.857
J48	0.934	0.955	...	0.753
Adaboost (DecisionStumps)	0.695	0.696	...	0.353
y45	0.670	0.670	...	0.116
Forward Weighting	0.994	1.000	0.114	0.980
Validierung	0.878	0.954	0.414	0.632

5. Wiederholung	ACC	ROCA	CXE	SLQ
Adaboost (J48)	0.961	0.961	...	0.854
Adtree	0.745	0.811	...	0.301
Bayboost	0.966	0.966		0.858
J48	0.934	0.956		0.743
Adaboost (DecisionStumps)	0.725	0.725	...	0.149
y45	0.574	0.605	...	0.048
Forward Weighting	0.968	0.999	0.269	0.901
Validierung	0.860	0.931	0.474	0.613

Da das Validierungsset jetzt überflüssig war, wurden die Modelle und der Gewichtsvektor neu gelernt um dann im Validierungsschritt das Validierungs-Sample durch die original Testdaten zu ersetzen und die endgültige Abgabe zu erstellen. Dazu wurden wieder fünf Samples gezogen, diesmal aber aus den gesamten 50.000 vorverarbeiteten Trainingsinstanzen und statt 30.000 wurde die Sample-Größe 35.000 gewählt. Das Gewicht wurde schließlich auf allen Trainingsdaten gelernt. Um die endgültige Vorhersage für das Test-Set zu erstellen wurde in der Validierungskette das Validierungs-Sample einfach durch das TestSet ersetzt, so dass sich die endgültigen Vorhersagen als die Summe der gewichteten Vorhersagen der Modelle für Instanz i geteilt durch die Anzahl der Gewichte ungleich Null ergab, also:

$$pred(i) = \frac{\sum_{k=1}^{30} PredModel_k(i) * Gewicht_k}{\sharp(Gewichte \neq Null)}$$

17.6.2 Die Abgabe

Da wiederum die Berechnung der Bayboost-Modelle am längsten dauerte, wurden während der Wartezeit bereits testweise Abgaben für alle Bewertungskriterien durch Ensembles ohne Bayboost-Modelle erstellt. Bis auf die Abgabe für slq wurden diese dann wieder verworfen. Der Grund dafür war, dass durch die Hinzunahme der Bayboost-Modelle zu dem Ensemble der Maximalwert der Vorhersagen (bei slq) auf 0.598433 (gegenüber 0.662772 beim Ensemble ohne Bayboost) beschränkt wurde und sich so die vermuteten positiven Vorhersagen stark in einem sehr kleinen Intervall von 0.5 bis 0.598433 stauten. In den drei anderen Fällen kam es zu keiner solchen Stauung und zudem verbesserte die Hinzunahme das Verhältnis zwischen positiven und negativen Vorhersagen in der Hinsicht, dass es der angenommen 50:50 Verteilung im Test-Set näher kam. Hier die Statisten der endgültigen Abgaben, die mittels R erstellt wurden:

Accuracy (ACC)

- Anzahl der Vorhersagen die kleiner als der von uns angegebenen threshold von 0.5 waren: 51.129
- Anzahl der Vorhersagen die größer oder gleich dem von uns angegebenen threshold von 0.5 waren : 48.871
- Wertebereich der Vorhersagen:
  - Min. : 0.02146
  - 1st Qu.: 0.13431
  - Median : 0.52380
  - Mean : 0.49786
  - 3rd Qu.: 0.86784
  - Max. : 0.93222
- Der auf dem Trainingsset erreichte Wert betrug 0.992 mit folgendem Gewichtsvektor:
  - adaboost.model1: 0.8
  - adaboost.model2: 1
  - adaboost.model3: 1
  - adaboost.model4: 0.999
  - adaboost.model5: 1
  - bayboost.model1: 0.95
  - bayboost.model2: 1
  - bayboost.model3: 0.97
  - bayboost.model4: 0.999
  - bayboost.model5: 1
  - adtree.model2: 1
  - adtree.model5: 0.65

Mean Cross Entropy (CXE)

- Anzahl der Vorhersagen die kleiner als der von uns angegebenen threshold von 0.5 waren: 52.723

- Anzahl der Vorhersagen die größer oder gleich dem von uns angegebenen threshold von 0.5 waren : 47.277
- Wertebereich der Vorhersagen:
  - Min. : 0.01461
  - 1st Qu.: 0.12702
  - Median : 0.55410
  - Mean : 0.52699
  - 3rd Qu.: 0.93891
  - Max. : 0.98889
- Der auf dem Trainingsset erreichte Wert betrug 0.126 mit folgendem Gewichtsvektor:
  - adaboost.model1: 1
  - adaboost.model2: 1
  - adaboost.model3: 1
  - adaboost.model4: 1
  - adaboost.model5: 1
  - bayboost.model1: 1
  - bayboost.model2: 1
  - bayboost.model3: 1
  - bayboost.model4: 1
  - bayboost.model5: 1
  - adtree.model4: 1

Area Under the ROC Curve (ROCA)

- Anzahl der Vorhersagen die kleiner als der von uns angegebenen threshold von 0.5 waren: 52.840
- Anzahl der Vorhersagen die größer oder gleich dem von uns angegebenen threshold von 0.5 waren : 47.160
- Wertebereich der Vorhersagen:
  - Min. : 8.950e-10

  - 1st Qu.: 7.932e-02
  - Median : 5.729e-01
  - Mean : 5.287e-01
  - 3rd Qu.: 9.933e-01
  - Max. : 9.978e-01

- Der auf dem Trainingsset erreichte Wert betrug 1.0 mit folgendem Gewichtsvektor:

  - bayboost.model1: 1
  - bayboost.model2: 1
  - bayboost.model3: 1
  - bayboost.model4: 1
  - bayboost.model5: 1

SLAC-Q Score (SLQ) (ohne Bayboost-Modelle)

- Anzahl der Vorhersagen die kleiner als der von uns angegebenen threshold von 0.5 waren: 41.529
- Anzahl der Vorhersagen die größer oder gleich dem von uns angegebenen threshold von 0.5 waren : 58.471
- Wertebereich der Vorhersagen:

  - Min. : 0.006131
  - 1st Qu.: 0.020642
  - Median : 0.388696
  - Mean : 0.351320
  - 3rd Qu.: 0.650303
  - Max. : 0.662772

- Der auf dem Trainingsset erreichte Wert betrug 0.973 mit folgendem Gewichtsvektor:

  - adaboost.model1: 0.75
  - adaboost.model2: 1
  - adaboost.model3: 0.75

- adaboost.model4: 0.5
- adaboost.model5: 0.75
- adtree.model2: 0.25

17.7 Ergebnisse des Wettbewerbs

Obwohl unser Ensemble-Ansatz gute Ergebnisse versprach, schnitten wir im Wettbewerb unterdurchschnittlich schlecht ab. Wir belegten insgesamt den 43. von 64 Plätzen. Auch in der Rangfolge der einzelnen Bewertungsmaße waren wir relativ schlecht. So wurden wir 42. bei der Optimierung der Accuracy, 43. im Maximieren der Fläche unter der Kurve (AUC) und 40. beim Minimieren der Kreuzentropie.

Group	Platz	ACC	Platz	ROCA	Platz	CXE	Gesamt
21	2.0	0.73187	1.0	0.83054	1.0	0.70949	1.333
425	1.0	0.73255	2.0	0.82754	2.0	0.72456	1.667
566	3.0	0.72775	3.0	0.82250	4.0	0.73001	3.333
532	5.0	0.72744	6.0	0.81791	3.0	0.72798	4.667
112	4.0	0.72745	4.0	0.82109	8.0	0.74371	5.333
421	6.0	0.72522	5.0	0.81906	6.0	0.73634	5.667
349	7.0	0.72424	8.0	0.81412	7.0	0.74137	7.333
24	11.0	0.72060	7.0	0.81572	5.0	0.73583	7.667
191	9.0	0.72304	9.0	0.81252	10.0	0.74632	9.333
14	8.0	0.72332	10.0	0.81208	14.0	0.75432	10.667
129	12.0	0.71884	11.0	0.81116	13.0	0.75301	12.000
449	15.0	0.71836	13.0	0.80952	11.0	0.74999	13.000
592	13.0	0.71883	12.0	0.81028	15.0	0.75644	13.333
159	14.0	0.71870	15.0	0.80829	12.0	0.75015	13.667
408	21.0	0.71584	16.0	0.80699	18.0	0.75878	18.333
64	16.0	0.71833	25.0	0.80310	16.0	0.75789	19.000
415	19.0	0.71790	21.0	0.80456	20.0	0.76745	20.000
584	20.0	0.71633	24.0	0.80419	22.0	0.77181	22.000
196	18.0	0.71824	20.0	0.80458	29.0	0.78607	22.333
44	26.0	0.71357	23.0	0.80428	19.0	0.76196	22.667
7	33.0	0.71096	31.0	0.79646	9.0	0.74423	24.333
167	25.0	0.71359	22.0	0.80445	27.0	0.78032	24.667
347	10.0	0.72196	14.0	0.80934	52.0	1.742E73	25.333
362	30.0	0.71234	26.0	0.80211	24.0	0.77623	26.667
182	27.0	0.71311	17.0	0.80642	38.0	0.86928	27.333
433	31.0	0.71141	27.0	0.79860	26.0	0.77967	28.000
27	32.0	0.71139	29.0	0.79779	28.0	0.78481	29.667
585	34.0	0.70972	30.0	0.79748	25.0	0.77690	29.667
577	45.0	0.69863	28.0	0.79796	17.0	0.75798	30.000
382	36.0	0.70663	32.0	0.79420	23.0	0.77459	30.333
66	37.0	0.70588	35.0	0.79074	21.0	0.76813	31.000
3	22.0	0.71560	19.0	0.80535	60.0	8.100E73	33.667

Group	Platz	ACC	Platz	ROCA	Platz	CXE	Gesamt
586	23.0	0.71544	18.0	0.80582	61.0	9.000E73	34.000
113	39.0	0.70511	37.0	0.78783	31.0	0.79766	35.667
8	41.0	0.70428	36.0	0.78959	30.0	0.79063	35.667
117	40.0	0.70508	34.0	0.79257	34.0	0.83395	36.000
321	29.0	0.71238	33.0	0.79303	50.0	1.449E73	37.333
500	43.0	0.70299	38.0	0.78413	32.0	0.80406	37.667
60	38.0	0.70577	40.0	0.77886	37.0	0.85489	38.333
317	17.0	0.71832	48.0	0.71813	53.0	2.535E73	39.333
138	46.0	0.69766	41.0	0.77626	35.0	0.83694	40.667
68	51.0	0.68778	39.0	0.78269	33.0	0.80625	41.000
⇒555	**42.0**	**0.70426**	**43.0**	**0.77386**	**40.0**	**0.98868**	41.667
26	24.0	0.71438	49.0	0.71429	54.0	2.571E73	42.333
276	49.0	0.69057	44.0	0.76975	39.0	0.87045	44.000
13	28.0	0.71304	51.0	0.71284	55.0	2.583E73	44.667
42	55.0	0.66814	47.0	0.73973	36.0	0.85084	46.000
572	52.0	0.68649	46.0	0.75629	43.0	1.14441	47.000
426	50.0	0.69006	45.0	0.76322	49.0	1.317E73	48.000
219	48.0	0.69284	54.0	0.63626	47.0	1.63563	49.667
409	54.0	0.67311	50.0	0.71333	45.0	1.18190	49.667
153	44.0	0.70010	52.0	0.69997	56.0	2.699E73	50.667
407	53.0	0.68107	53.0	0.68107	51.0	1.656E73	52.333
518	61.0	0.57431	55.0	0.63547	44.0	1.14744	53.333
148	62.0	0.53276	59.0	0.54706	42.0	1.04035	54.333
154	59.0	0.58207	58.0	0.59276	46.0	1.37861	54.333
352	64.0	0.29094	62.0	0.31402	41.0	1.00119	55.667
187	63.0	0.49942	61.0	0.50224	48.0	1.170E73	57.333
351	57.0	0.62920	57.0	0.62918	58.0	3.337E73	57.333
142	58.0	0.61752	56.0	0.63344	59.0	3.714E73	57.667
264	60.0	0.57565	60.0	0.52715	57.0	3.264E73	59.000
318	35.0	0.70858	-	-	-	-	-
385	47.0	0.69455	42.0	0.77415	-	-	-
568	56.0	0.64672	-	-	-	-	-

Tabelle 17.5: Platzierungen der Wettbewerbsteilnehmer (Ergebnisse der Projektgruppe sind markiert)

17.7.1 Resultate der einzelnen Bewertungsmaße

Accuracy

Die höchste Accuracy, die erreicht wurde, betrug 0.73187. Wir erreichten nur 0.70426. Bei der Ausarbeitung unserer Lösung war es jedoch ein Leichtes, eine Accuracy von über 0.8 zu erhalten. Die Frage ist nun, warum die Ergebnisse für den Testdatensatz so eklatant von denen des Trainingsdatensatzes abwei-

chen. Hierauf werden wir eingehen, wenn wir die Ergebnisse für die anderen Bewertungsmaße abgehandelt haben.

Area under the curve

Für Area under the curve betrug der beste Wert 0.83054. Wir kamen nur auf 0.77386. Auch hier waren die Werte in der Bearbeitung des Trainingsdatensatzes viel besser als beim Testdatensatz.

Kreuzentropie

Leider konnten wir auch bei Kreuzentropie keine zufriedenstellenden Ergebnisse erzielen. 0.98868 im Gegensatz zu 0.70949 ist schon ein eklatanter Unterschied.

17.7.2 Überlegungen zu den schlechten Ergebnissen

Das konstant schlechte Abscheiden unserer Gruppe überrascht doch etwas. Vor allem, da - wie gesagt - die kreuzvalidierten Ergebnisse auf dem Trainingsdatensatz allesamt gut waren.
Erste Überlegungen gingen natürlich direkt in Richtung einer falschen Stichprobe. Hatten wir vielleicht nicht-disjunkte Stichproben gezogen?
Obwohl wir sicher waren, diesen Fehler nicht gemacht zu haben, wurden nochmal zwei disjunkte Stichproben aus dem Trainingsdatensatz gezogen. Mit einer dieser Stichproben wurde ein Adaboost mit J48 gelernt, dessen Modell auf die zweite Stichprobe angewandt wurde. Eine Accuracy von über 85% zerstreute unsere Bedenken hinsichtlich einer nicht-disjunkten Stichprobe.
Die zweite und wahrscheinlichere Fehlerursache liegt in unserer Bearbeitung der fehlenden Werte. Hatten wir hier einen Fehler gemacht?
Die Tabellenspalten, die fehlende Werte enthielten, wurden ja mit sinnvollen (gelernten) Werten aufgefüllt. Die Tabellenspalten im Testdatensatz wurden mit auf Testdaten gelernten Werten aufgefüllt. Beim Trainingsdatensatz waren die Werte natürlich auf dem Trainingsdatensatz gelernt.
Leider ist uns erst jetzt aufgefallen, das beim Lernen der Werte des Trainingsdatensatzes das gegebene Label mit als Attribut verwendetet wurde, was nun höchstwahrscheinlich das Problem aufwirft, dass aufgrund einer möglichen Korrelation zwischen gelernten Werten und Label die Klasseninformation in

die gelernten Attributwerte hineinkodiert wurde. Im Falle des Testdatensatzes standen zum Lernen der Attributwerte die Label dann natürlich nicht mehr zur Verfügung.
Wir haben unser Verfahren noch einmal angewandt, wobei diesmal die Attribute, die fehlende Werte enthielten, komplett aus den Daten entfernt wurden. Diese neuen Ergebnisse bestätigen das schlechte Abschneiden. Auf einem Validierungsdatensatz der Größe 10.000 wurden die folgenden Ergebnisse erziehlt:

	ACC	AUC	CXE	SLQ
Adaboost (J48)	~ 0.68	~ 0.68	-	~ 0.22
Adtree	~ 0.69	~ 0.77	-	~ 0.23
J48	~ 0.67	~ 0.68	-	~ 0.21
Adaboost (DecisionStumps)	~ 0.32	~ 0.32	-	~ 0.14
y45	~ 0.69	~ 0.76	-	~ 0.24
Forward Weighting(Trainingsfehler)	0.916	0.971	0.438	0.754
Validierung	0.698	0.781	0.789	0.256

Diese Ergebnisse bestätigen die von der Gruppe erreichten Endplazierungen. Weiterhin ist zu bemerken, daß das falsche Vorgehen der Gruppe bei der Ergänzung der fehlenden Werte offensichtlich nicht den Lernverfahren an sich geschadet hat, sondern nur falsche Qualitätsmaße lieferte. So ist zum Beispiel die tatsächlich erreichte Accuracy rund 15% geringer als der Wert, der durch das Lernen auf den Daten mit den ergänzten Werte zustande kam. Abschließend bleibt zu sagen, dass der Ansatz so falsch nicht war, allerdings hätte die Gruppe versucht, diesen Ansatz noch weiter zu verbesseren, wenn bekannt gewesen wäre, wie schlecht die Ergebnisse tatsächlich waren. So wären z.B. mit Sicherheit die geboosteten Entscheidungsstümpfe aus dem Ensemble herausgenommen worden und sattdessen wären andere Modelltypen aufgenommen worden. Ebenso konnten cxe-Werte unter 1 nur durch ein Ensemble erreicht werden und einige der Abgaben der anderen Teilnehmer bestätigen diese Aussage (der letzte Platz hatte einen cxe-Wert von $9 * 10^{73}$).

Kapitel 18

Protein-Homologie-Aufgabe

Dirk Dach, Holger Flick, Christophe Foussette, Daniel Hakenjos, Christian Kullmann, Siehyun Strobel

18.1 Aufgabenstellung

18.1.1 Datenumfang

Bei der Protein Homologie Aufgabe sollte vorhergesagt werden, welche Proteinsequenzen homolog zu einer Ursprungssequenz sind. Zu jeder Ursprungssequenz gab es ca. 1000 Proteinsequenzen, für die Homologie vorhergesagt werden sollte. Positive Beispiele waren homolog negative Beispiele nicht homolog. Eine genauere Analyse der Datensätze findet sich in Abschnitt 18.2. Über die Daten war nichts bekannt außer, daß sie aus dem Bereich der Bioinformatik stammen. Die Herkunft der Daten wurde erst nach dem Ende des KDD-Cups bekanntgegeben.

18.1.2 Performanzmaße

Die Aufgabenstellung bestand darin Modelle zu finden, um die folgenden vier Performanzmaße zu optimieren.

TOP1

Die Beispiele werden blockweise nach dem Wert der Vorhersage sortiert. Für das am höchsten eingestufte Beispiel wird überprüft, ob es homolog zur Ursprungssequenz ist. Falls ja erhält man einen Punkt für diesen Block, ansonsten null Punkte. Die Punkte werden zusammengerechnet und durch die Gesamtzahl der Blöcke dividiert. TOP1 sollte maximiert werden. Der höchste zu erreichende Wert ist Eins, der schlechteste Null.

RKL (Ranking Last)

Beim RKL wird der durchschnittliche Rang des letzten homologen Beispiel gemessen. Dazu werden die Beispiele wieder nach dem Wert der Vorhersage sortiert. Für jeden Block wird der Rang des homologen Beispiels ermittelt, daß am weitesten unten steht und der Durchschnitt über alle Blöcke bestimmt. RKL sollte minimiert werden. Da in einigen Blöcken mehr als ein homologes Beispiel war, konnte der Idealwert von Eins nicht erreicht werden.

Average Precision

Für die Average Precison wurde aus den vielen Definitionen eine etwas ungewöhnliche benutzt, die manchmal auch 'Expected Precision' genannt wird. Precision ist der Anteil von positiv vorhergesagten Beispielen, über einem bestimmten Schwellwert, die auch wirklich positiv sind. Der Schwellwert wird verändert, so daß eins nach dem anderen jedes Beispiel positiv klassifiziert wird. Dies hat Ähnlichkeit mit dem 'Area Under the Curve'-Maß, ist aber nicht vollkommen identisch dazu. Für jeden Schwellwert wird die Precision und von allen der Durchschnitt berechnet. Für den Fall von Gleichständen zwischen positiven und negativen Beispielen wir die Average Precision aller möglichen Anordnungen gebildet. Zum Schluß wird der Durchschnitt über alle Blöcke gebildet. Die Average Precision sollte maximiert werden. Der Idealwert

beträgt Eins, der schlechteste Wert Null.

RMSE (Root-Mean-Squared-Error)

Zur Berechnung des Root-Mean-Squared-Error wird die übliche Definition benutzt. Sei c die wahre Klasse des Beispiels, p die getroffene Vorhersage und n die Anzahl der Beispiele. Dann wird der RMSE folgendermaßen berechnet:

$$\sqrt[2]{\frac{1}{n}\sum (c-p)^2}$$

Es wird also für jedes Beispiel die Differenz von wahrer Klasse und Vorhersage gebildet und das Ergebnis quadriert. Dies wird über alle Beispiele gemittelt und abschließend die Quadratwurzel gezogen.

18.2 Dateninspektion

Vor Erarbeitung von Lösungsansätzen ist es notwendig, die Daten im Detail zu kennen. Dazu ist es zunächst erforderlich die Angaben des Herausgebers darzustellen und diese nachzuvollziehen. Anschließend kann eine Untersuchung der Verteilung der einzelnen Werte erfolgen. Dazu wurde die Statistik-Software 'R' und das Data Mining Tool 'Mining Mart' zu Hilfe genommen.

18.2.1 Beschreibung der Rohdaten

Der Veranstalter veröffentlichte 2 Dateien: Eine Datei mit Datensätzen zum Trainieren, die andere mit Daten zum Testen. Die Struktur der beiden Dateien war identisch.

Struktur

Jede Zeile der Daten beschreibt genau ein Beispiel (vgl. Tabelle 18.2). Die einzelnen Attribute sind durch ein Tabulatorzeichen voneinander getrennt. Das erste Element in jeder Zeile ist die sogenannte 'Block-ID', die angibt, zu welcher Sequenz ein Beispiel gehört. Für jede Sequenz gibt es eine eindeutige Block-ID,

Set	Wert	Bedeutung
Training	0	nicht-homologes Beispiel ('decoy'), negativ
	1	homologes Beispiel, positiv
Test	?	Label wird nicht weggelassen, sondern mit '?' aufgeführt.

Tabelle 18.1: Ausprägung des Labels in Trainings- und Testdatensatz

Block ID	Unique ID	Label	Attribute (*feature values*)				
			x_1	x_2	...	x_{73}	x_{74}
279	261532	0	52.00	32.69	...	−0.350	0.26

Tabelle 18.2: Übersicht über den Aufbau eines Beispiels

wobei in der Aufgabenstellung darauf hingewiesen wird, daß diese IDs nicht fortlaufend sein müssen. Zusätzlich wird jedes Beispiel mit einer eindeutigen 'Example ID' gekennzeichnet, die an zweiter Position zu finden ist. Das Label, welches die Klasse des Beispiels angibt, steht an der dritten Position. Es gibt für das Label drei mögliche Ausprägungen, die in Tabelle 18.1 aufgelistet sind.

Bei allen folgenden 74 Attributen handelt es sich um Gleitkommazahlen, die in der Aufgabenstellung als 'feature values' bezeichnet werden. Diese Werte drücken die Übereinstimmung zwischen der nativen Protein-Sequenz und der auf Homologie getesteten Sequenz aus. Genauere Angaben sind nicht veröffentlicht worden.

In der Aufgabenstellung wird zuletzt noch darauf hingewiesen, daß es in beiden Rohdaten-Dateien keine fehlenden Werte gibt, d.h. die oben beschriebene Struktur wird für jedes Beispiel genau eingehalten. Wir haben die Vollständigkeit der Daten jedoch sicherheitshalber überprüft, wobei wir bestätigen konnten, daß es keine fehlenden Werte in den Datensätzen gibt.

Anzahl	Training	Test
Beispiele	145.751	139.658
Blöcke	153	150
positive Beispiele	1.296 (0,889%)	-

Tabelle 18.3: Übersicht über den Datenumfang

Umfang

Unser erster Schritt war die grobe Sichtung der Rohdaten. Hierzu wurden die Daten zunächst in eine Oracle Datenbank eingelesen. Im ersten Semester haben wir dazu entsprechende Tools entwickelt (vgl. Anhang), die ein schnelles und umkompliziertes Einlesen ermöglicht haben.

Mit Hilfe von SQL-Abfragen zählten wir die gegebenen Beispiele in beiden Datensätzen, ermittelten die Anzahl der Blöcke und verschafften uns einen Eindruck von der Menge an positiv klassifizierten Sequenzen. Tabelle 18.3 fasst diese Informationen zusammen. Es fällt sofort auf, daß der Anteil an positiven Beispielen sehr gering ist.

Wir versuchten nun, wie im nächsten Abschnitt beschrieben, die Verteilung der 'feature values', im Folgenden kurz mit 'Attribute' bezeichnet, zu analysieren. Dabei wollten wir zunächst sicherstellen, daß die Verteilung der Attribute in Trainings- und Testdatenmenge identisch ist. Zudem hofften wir darauf, Abhängigkeiten zu finden, die uns auf mögliche Lösungsansätze verweisen. Als Notation für die Attribute verwendeten wir die Bezeichnung x_1 bis x_{74}.

18.2.2 Verteilung der Attribute

Wie in Abschnitt 18.2.1 erwähnt, wurden vom Veranstalter zwei Dateien zur Verfügung gestellt. Um für die spätere Validierung der Lernergebnisse sicherzugehen, daß die Daten im Validierungsset von der gleichen Art sind wie die Daten in der Trainingsmenge, haben wir die Verteilungen untersucht.

Dazu haben wir die Daten in das Statistikprogramm R eingelesen und Histogramme für alle 74 Attribute erstellt. Dies wurde sowohl für die Trainings-, als auch für die Testmenge durchgeführt. Anschließend konnten wir die Histogramme vergleichen. Die Abbildungen 18.1-18.8 sind eine kleine Auswahl der Histogramme aller 74 Attribute. Bei den Abbildungen ist jeweils links die Verteilung der Trainingsmenge und rechts die Verteilung der Testmenge zu sehen. Zusätzlich sind in den Histogrammen der Trainingsmenge die positiv kategorisierten Beispiele rot gekennzeichnet. Es ist mit Hilfe der Abbildungen sehr klar und verständlich zu sehen, daß die beiden Datenmengen nahezu eine identische Verteilung aufweisen. Weiterhin spiegelt sich der geringe Anteil der positiven Beispiele wieder. Abschnitt 18.2.5 untersucht die Anzahl der positiven Beispiele bezogen auf die einzelnen Blöcke (die über die Block-ID gebildet

werden können).

18.2.3 Abhängigkeiten zwischen Label und Attributen

Nach der allgemeinen Betrachtung der Verteilungen haben wir versucht, einen Bezug zwischen den einzelnen Attributen und dem Label zu finden. Dazu erstellten wir erneut 74 Plots, die das jeweilige Attribut auf der x-Achse gegen das Label auftragen.

Wir erhofften uns eine visuelle Einteilung der verschiedenen Attribute in eine übersichtliche, handliche Anzahl von Klassen. Die Plots ergaben jedoch keinerlei Beziehungsmuster zwischen Label und den einzelnen Attributen. Mit einigen Bemühungen konnte man zwischen 6 Gruppen unterscheiden, jedoch ergaben diese Abgrenzungen keinerlei Lösungsansatz für die Wahl eines möglichen Lernverfahrens bzw. für die weitere Datenaufbereitung.

Abbildung 18.9 zeigt die Plots für die Attribute x_{11} und x_{12}. Sie zeigt zwei mögliche 'Gruppen', die visuell entstehen. Trotzdem war uns eine formale Abgrenzung nicht möglich.

18.2.4 Untersuchung der Blöcke

Uns erschien die Betrachtung der gesamten Daten zu grob und wir entschieden uns daher, die Attribute für jeden einzelnen der 153 Blöcke in den Trainingsdaten zu untersuchen. Dazu ermittelten wir mittels SQL-Anfragen für jedes der 74 Attribute

- Minimum und Maximum
- Varianz bzw. Standardabweichung
- Mittelwert bzw. Median

Es war uns danach zwar immernoch nicht möglich eine manuelle Unterscheidung der Blöcke vorzunehmen, da die ermittelten Werte sehr stark variierten und kein Muster erkennen ließen, jedoch brachte dieser Ansatz uns auf den Lösungsansatz der Blocknormierung bzw. der Klassifikation der Blöcke mittels

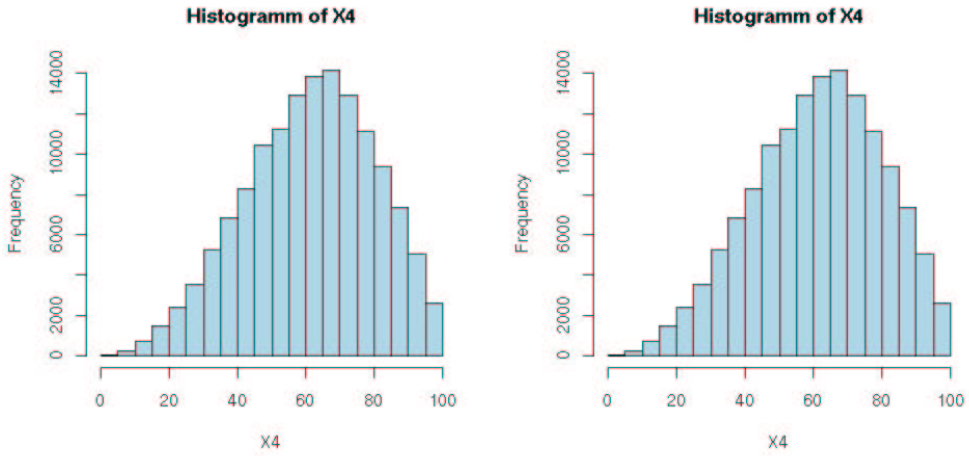


Abbildung 18.1: Verteilung der Werte für Attribut 1 (x_1); Trainingsdaten links, Testdaten rechts

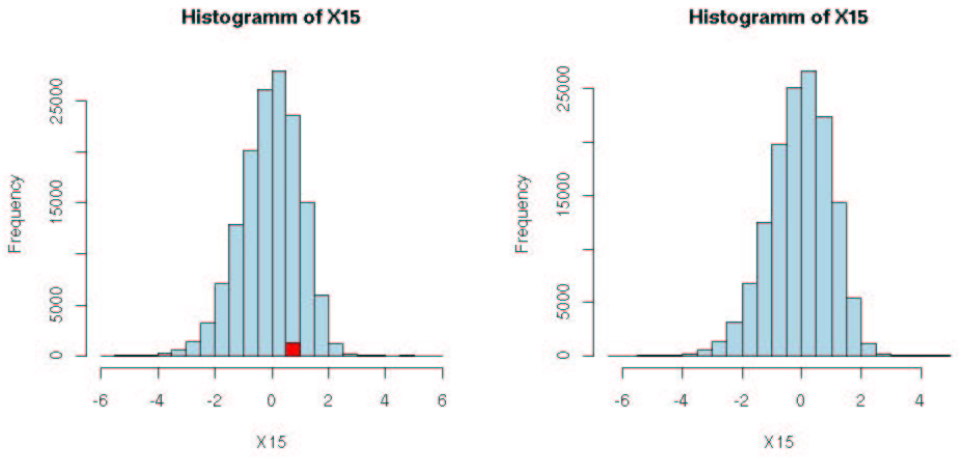


Abbildung 18.2: Verteilung der Werte für Attribut 12 (x_{12}); Trainingsdaten links, Testdaten rechts

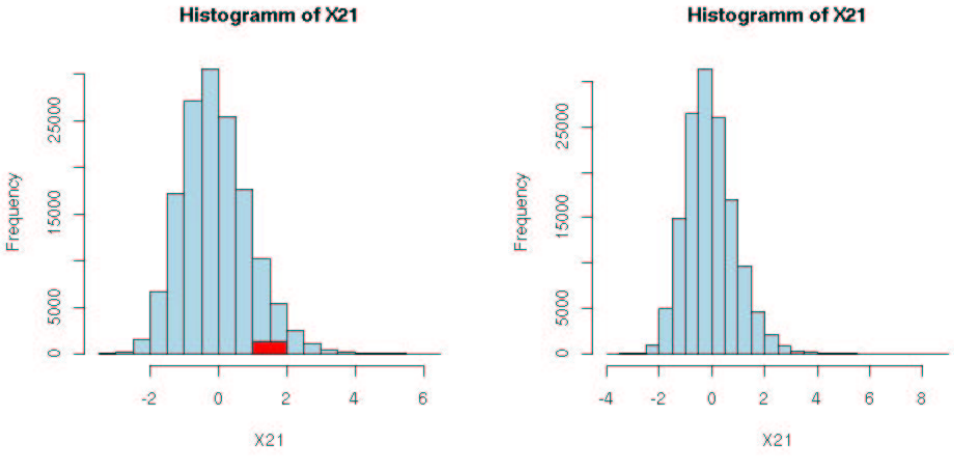


Abbildung 18.3: Verteilung der Werte für Attribut 18 (x_{18}); Trainingsdaten links, Testdaten rechts

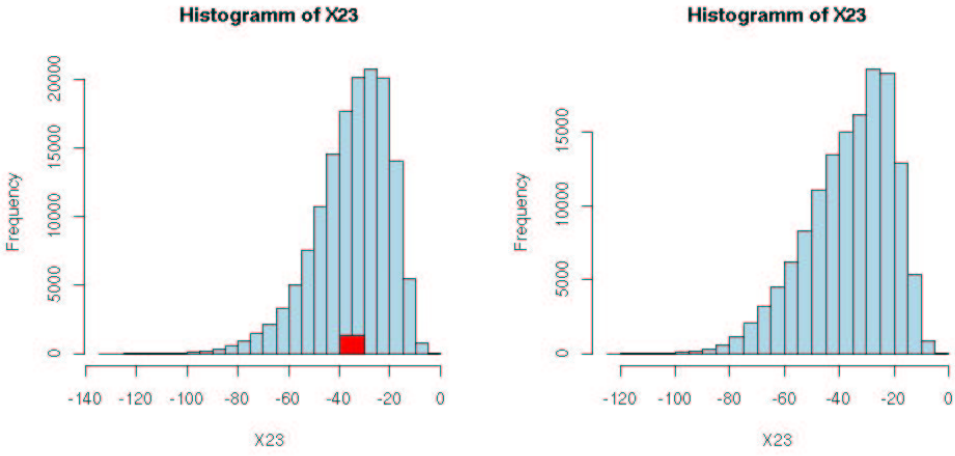


Abbildung 18.4: Verteilung der Werte für Attribut 20 (x_{20}); Trainingsdaten links, Testdaten rechts

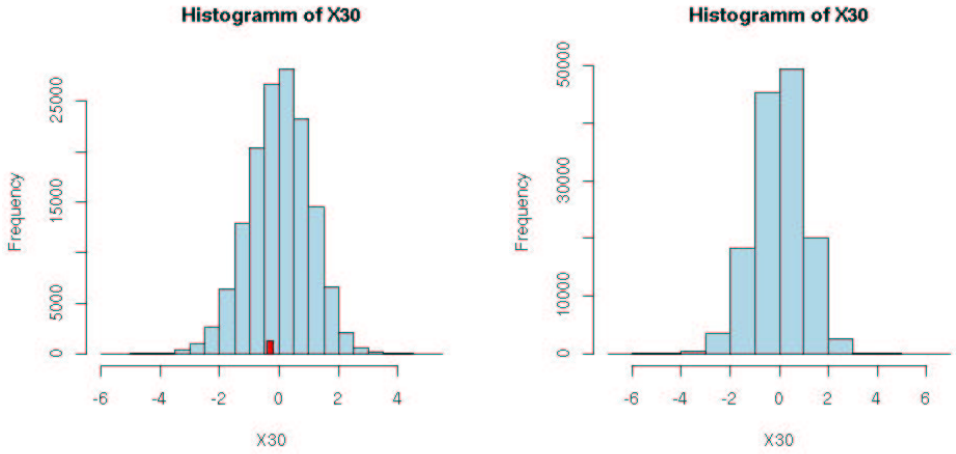


Abbildung 18.5: Verteilung der Werte für Attribut 27 (x_{27}); Trainingsdaten links, Testdaten rechts

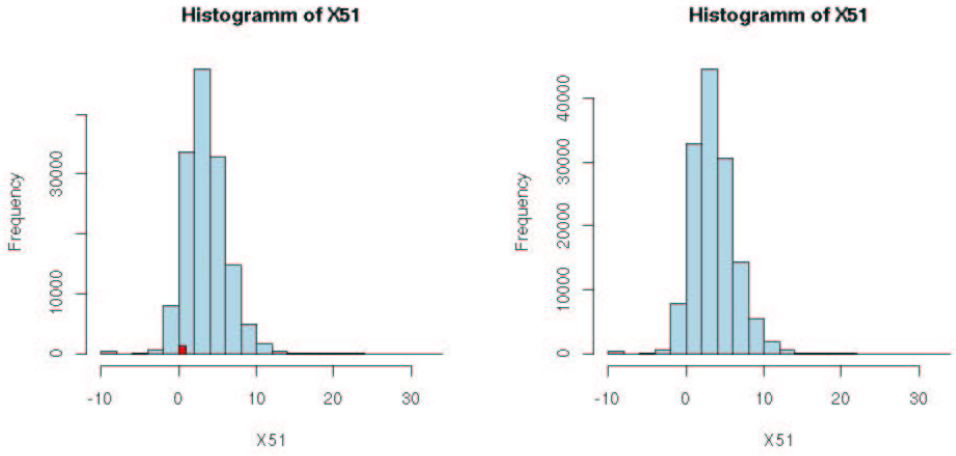


Abbildung 18.6: Verteilung der Werte für Attribut 48 (x_{48}); Trainingsdaten links, Testdaten rechts

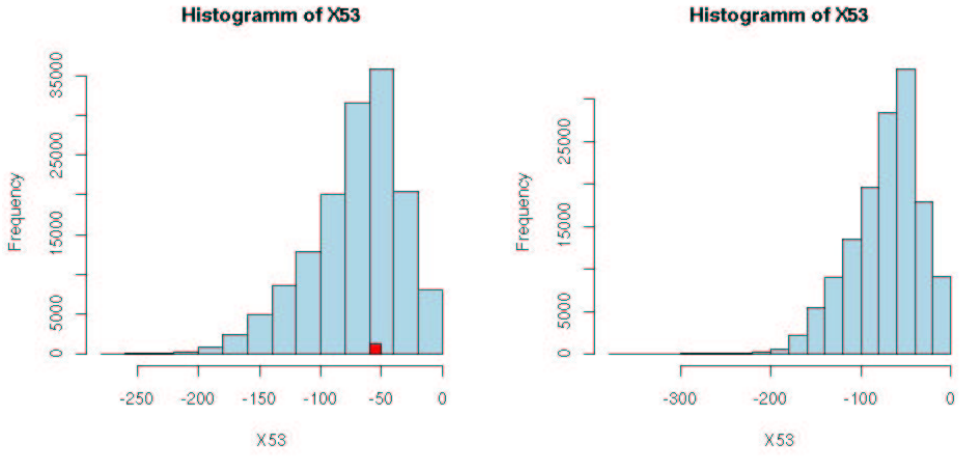


Abbildung 18.7: Verteilung der Werte für Attribut 50 (x_{50}); Trainingsdaten links, Testdaten rechts

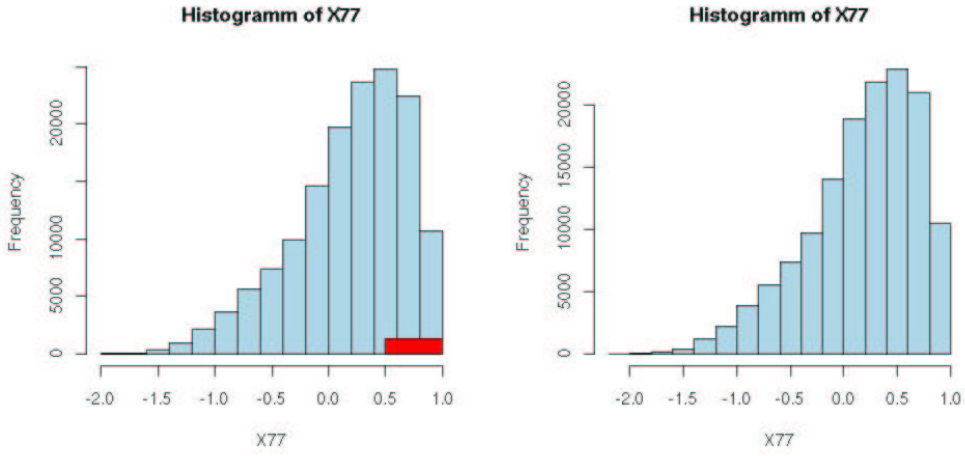


Abbildung 18.8: Verteilung der Werte für Attribut 74 (x_{74}); Trainingsdaten links, Testdaten rechts

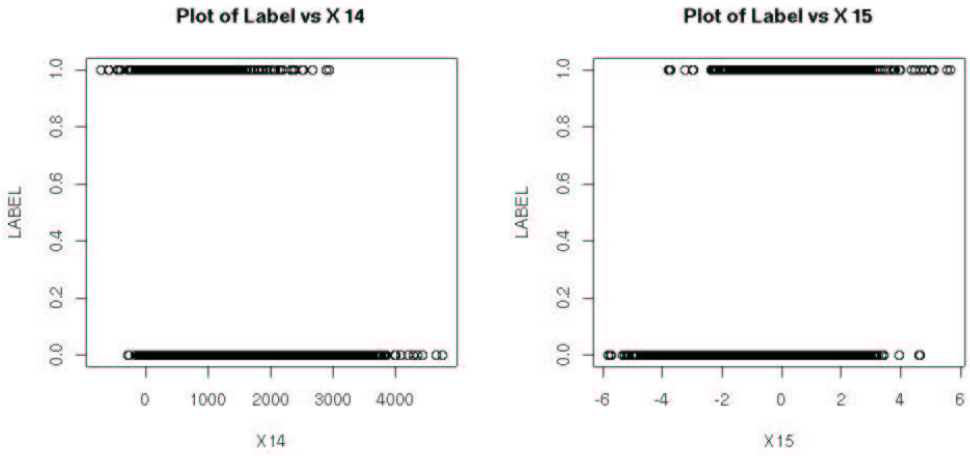


Abbildung 18.9: Attribute x_{11} und x_{12} abgetragen gegen das Label

dieser Eigenschaften. Diese Ansätze werden im Abschnitt 18.3 (Datenvorverarbeitung) näher erläutert.

Weiterhin war es unser Ziel, Lösungsansätze mit Hilfe von Blöcken mit zahlreichen positiven Beispielen zu erzielen.

18.2.5 Verteilung der positiven Label bezogen auf die Blöcke

Um unsere Lernverfahren mit möglichst guten Stichproben versorgen zu können, waren wir bestrebt, Blöcke auszuwählen, die viele positive Beispiele enthalten. Dazu erstellten wir mittels SQL eine Übersicht, die die Anzahl der positiven Beispiele für jeden Block auflistet. Diese Tabelle war unser Ausgangspunkt für weitere Schritte der Datenvorverarbeitung, der Erarbeitung und Auswahl geeigneter Lernverfahren.

Block	# Positive	# Block	%	Block	# Positive	# Block	%
2	4	995	0.402	5	2	1059	0.189
7	23	975	2.359	8	2	625	0.320
9	1	911	0.110	12	24	958	2.505
13	12	1101	1.090	14	2	906	0.221
16	13	993	1.309	17	1	1157	0.086
18	1	1114	0.090	19	3	1111	0.270
23	8	935	0.856	24	1	929	0.108
25	24	817	2.938	27	7	820	0.854
29	5	988	0.506	30	1	749	0.134

Block	# Positive	# Block	%	Block	# Positive	# Block	%
31	4	1076	0.372	33	2	978	0.204
34	1	1143	0.087	36	26	826	3.148
37	1	980	0.102	39	1	1194	0.084
42	1	981	0.102	46	2	616	0.325
48	23	942	2.442	55	15	815	1.840
57	18	962	1.871	59	4	974	0.411
60	4	742	0.539	61	2	924	0.216
62	5	1018	0.491	64	3	635	0.472
65	11	884	1.244	66	1	1185	0.084
67	2	612	0.327	69	14	1099	1.274
73	8	1031	0.776	74	4	630	0.635
77	2	993	0.201	78	1	1001	0.100
80	5	944	0.530	81	3	1024	0.293
95	1	1120	0.089	96	2	974	0.205
100	28	796	3.518	101	5	793	0.631
103	7	1054	0.664	104	5	1005	0.498
106	1	964	0.104	107	1	1172	0.085
110	10	1132	0.883	111	1	993	0.101
112	31	934	3.319	113	6	677	0.886
114	2	845	0.237	115	21	761	2.760
116	19	969	1.961	117	11	1136	0.968
118	30	834	3.597	119	2	831	0.241
124	1	824	0.121	125	1	932	0.107
129	2	803	0.249	130	19	821	2.314
133	1	1090	0.092	135	16	1057	1.514
138	5	780	0.641	139	2	877	0.228
142	1	889	0.112	143	5	937	0.534
144	2	880	0.227	146	9	985	0.914
147	21	949	2.213	148	1	937	0.107
156	2	878	0.228	158	4	1128	0.355
159	1	848	0.118	160	2	1219	0.164
162	37	816	4.534	163	23	937	2.455
164	10	762	1.312	167	9	978	0.920
170	12	1036	1.158	172	9	926	0.972
173	1	1159	0.086	174	22	817	2.693
175	5	998	0.501	176	1	1048	0.095
178	25	1013	2.468	180	1	1244	0.080
182	4	833	0.480	184	15	1130	1.327
185	17	985	1.726	186	22	817	2.693
187	1	854	0.117	189	14	947	1.478
191	2	908	0.220	198	48	846	5.674
201	1	1073	0.093	205	7	1214	0.577
206	1	981	0.102	210	6	962	0.624
211	1	1059	0.094	212	9	746	1.206
214	2	804	0.249	216	3	1068	0.281
228	9	1086	0.829	230	3	910	0.330
231	12	935	1.283	232	1	930	0.108
234	4	859	0.466	236	2	1050	0.190
237	3	813	0.369	238	50	861	5.807
239	21	949	2.213	241	9	910	0.989
243	11	941	1.169	244	32	817	3.917

Block	# Positive	# Block	%	Block	# Positive	# Block	%
245	6	954	0.629	246	33	801	4.120
250	1	965	0.104	252	2	1077	0.186
254	6	949	0.632	255	7	1083	0.646
256	13	942	1.380	257	1	1127	0.089
259	3	1025	0.293	261	1	1058	0.095
262	3	930	0.323	264	5	1063	0.470
266	10	991	1.009	267	1	988	0.101
269	5	1134	0.441	270	2	1056	0.189
271	49	1006	4.871	273	3	1191	0.252
274	19	1011	1.879	275	4	934	0.428
276	6	811	0.740	277	24	917	2.617
279	1	805	0.124	280	1	1102	0.091
282	1	1032	0.097	283	1	700	0.143
286	5	1145	0.437	287	9	875	1.029
289	5	902	0.554	291	5	999	0.501
293	1	973	0.103	299	1	987	0.101
303	22	1020	2.157				

Abschließend soll Abbildung 18.10 graphisch verdeutlichen, daß die positiven Beispiele in den Blöcken meist nur in sehr geringer Anzahl auftreten. Weiterhin weisen die Blöcke mit vielen positiven Beispielen sehr unterschiedliche Häufigkeiten auf, was das Finden eines geeigneten Ansatzes nicht erleichterte.

18.3 Lösungsansatz

Datenvorverarbeitung

Einleitung Nachdem die Daten gesichtet wurden, mussten verschiedene Änderungen an der Formatierung und Skalierung der Daten vorgenommen werden. Das begründet sich hauptsächlich durch die Wahl der verwendeten Programme, die oft eine spezielle Formatierung der Eingabe-Daten verlangen. Aber es war auch notwendig, die Daten zu normieren, um bessere Lernergebnisse zu erzielen. Die verwendeten Vorgehensweisen werden im folgenden vorgestellt.

Einspielen in eine Datenbank Als erstes wurden sowohl die Trainings- als auch die Testdaten in eine Oracle Datenbank übertragen. Dies wurde mit dem bereits im Vorfeld von der Gruppe entwickelten Programm "Data2DB"

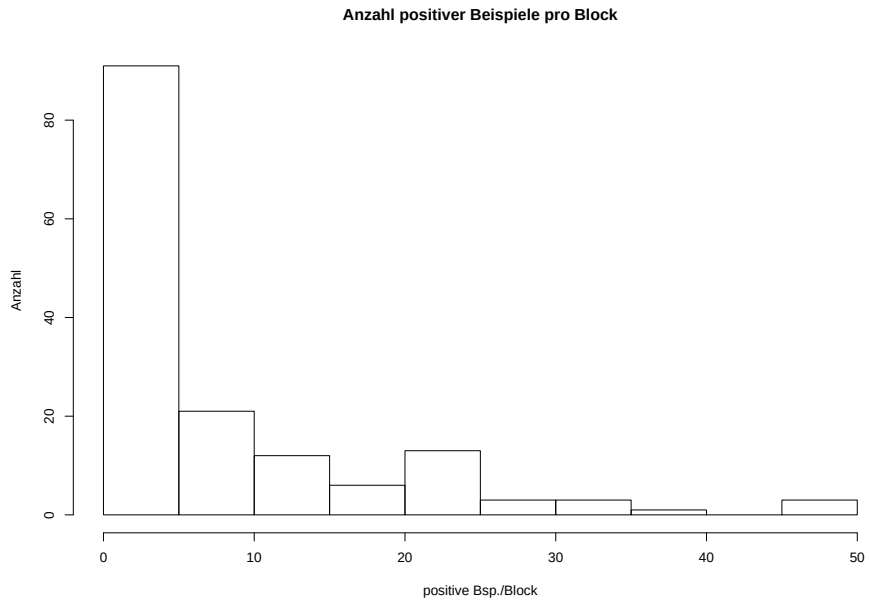


Abbildung 18.10: Häufigkeiten der pos. Beispiele pro Block

realisiert[1] (siehe Anhang A).

Formatieren für SVMLight Des weiteren wurden die Daten in ihrer ursprünglichen Fassung so umformatiert, dass sie durch das Programm SVM-Light bearbeitet werden konnten. Die Formatierung für das Programm sieht vor, dass der Wert des Zielattributs in der ersten Spalte steht und dabei den Wert +1, -1 oder 0 haben muss oder aber eine reelle Zahl ist. Als nächstes folgen die Werte der anderen Attribute eines Datensatzes, angeführt von der Attribut-Nummer. Als Trennzeichen zwischen Attributnummer und Attributwert dient hier ein Doppelpunkt. Letztlich sieht ein Zeileneintrag folgendermassen aus:

```
<zielattribut> <attribut>:<wert> ... <attribut>:<wert>
wobei
<zielattribut> .=. +1|-1|0|reelle Zahl
<attribut> .=. ganze zahl|'qid'
<wert> .=. reelle Zahl
```

[1]Hierbei sind keine Schwierigkeiten aufgetreten, da keine NULL-Werte in den Rohdaten vorhanden waren.

Normierung Um besser mit den Daten arbeiten zu können, und diese anschliessend mit dem zur Verfügung gestellten **PERF** bewerten zu können, wurden die Daten anschliessend normiert. Alle Attribute wurden auf das Intervall $[0, 1]$ normiert.

Normierung im Bezug zu unterschiedlichen Blöcken Bei der Normierung hinsichtlich der unterschiedlichen Blöcke wurde ähnlich verfahren, wie vorher beschrieben, jedoch wurden hier die Blöcke als eigenständige Einheiten betrachtet. Das bedeutet, dass die für die Normierung verwendeten Maximal- und Minimal-Werte jeweils die Maximal- und Minimal-Werte eines Attributs innerhalb eines Blockes waren.

Block2Vector Bei Block2Vector handelt es sich um ein Vorgehen, welches aus den einzelnen Blöcken unserer Ursprungsdaten jeweils einen Vector erstellt, dessen Werte aus den Minimal- und Maximal-Werten der Attribute bestanden, sowie aus dem Median und dem Durchschnitt. Diese Daten wurden dann als Eingabedaten bzw. zusätzliche Daten für die Lernverfahren verwendet.

18.3.1 Bewertungsmaß *Root Mean Squared (RMS)*

Erste Ansätze

Um ein erstes "Gefühl" für das Bewertungsmaß zu erhalten wurden einige Lernverfahren angestoßen. Insgesamt wurden auf den Daten sechs Verfahren angewendet:

- Bayboost mit Y45 als inneren Lerner
- Adaboost mit J48 als Lerner
- Logitboost mit Decisionstump als Lerner
- J48
- eine Support Vector Machine (SVM light) mit einem linearen Kernel
- und eine SVM light mit RBF-Kernel

Beim Lernen auf den gesamten Daten mit einer 10-fachen Kreuzvalidierung erzielten die einzelnen Verfahren zwar Ergebnisse unterschiedlicher Performanz, doch das Erscheinungsbild ähnelt sich sehr: Es werden bei allen Lernverfahren eine gute bis sehr gute Accuracy und Precision erzielt, doch kein Lerner kann den Raum der positiv-klassifizierten Beispiele ebenso gut eingrenzen, was sich unter anderem bei dem Recall bemerkbar macht.

Lernverfahren	Accuracy	Precision	Recall	RMS
Bayboost mit Y45	99,76%	95,11%	79,86%	0,049
Adaboost mit J48	99,77%	97,54%	76,62%	0,047
Logitboost mit Decisionstump	99,72%	99,62%	75,08%	0,049
J48	99,66%	99,76%	71,624%	0,055
SVM mit lin. Kernel	99,6%	99,2%	57,2%	0,0454
SVM mit RBF-Kernel	99,6%	99,1%	61,7%	0,0425

Tabelle 18.5: Ergebnisse der Lernverfahren bei einer 10-fachen Kreuzvalidierung auf den gesamten Daten

Bei den ersten Experimenten erzielte eine SVM bzgl. des *RMS* die besten Ergebnisse. Aus diesem Grund wurde unter *Yale* eine Parameteroptimierung durchgeführt, um die bereits vorhandenen Ergebnisse möglicherweise noch zu optimieren (für die zugehörige Yale-Kette siehe 18.11). Auf diesem Weg ließ sich der Gammawert des RBF-Kernels, sowie der C-Wert beider SVMs optimieren (der optimierte Gamma-Wert konnte auch bei der Bearbeitung der anderen Bewertungsmaße verwendet werden).

Mit diesen optimierten Parametern konnte tatsächlich auch der RMS verbessert werden. Der beste Wert erzielte dabei eine SVM mit RBF-Kernel (Gamma=0.189 und C=31): Der Root Mean Squared-Error verbesserte sich auf diese Weise von 0.041753 auf 0.036095. Jedoch blieb das Problem bestehen, dass die wenigen positiven Beispiele nicht zuverlässig genug richtig klassifiziert wurden. Das Ziel nachfolgender Untersuchungen war es also, die Zahl falsch-negativer Beispiele zu reduzieren.

Lösungsansatz

Um die Anzahl der falsch-negativen Beispiele zu reduzieren, kann neben einer SVM mit RBF-Kernel (die bisher das beste Ergebnis erzielte) noch ein zusätzliches Verfahren zum Lernen einbezogen werden, so dass auf diesem Weg mehr positive Beispiele richtig klassifiziert werden können. Die Vorhersagen beider Lernverfahren können mit einer einfachen *ODER*-Verknüpfung miteinander

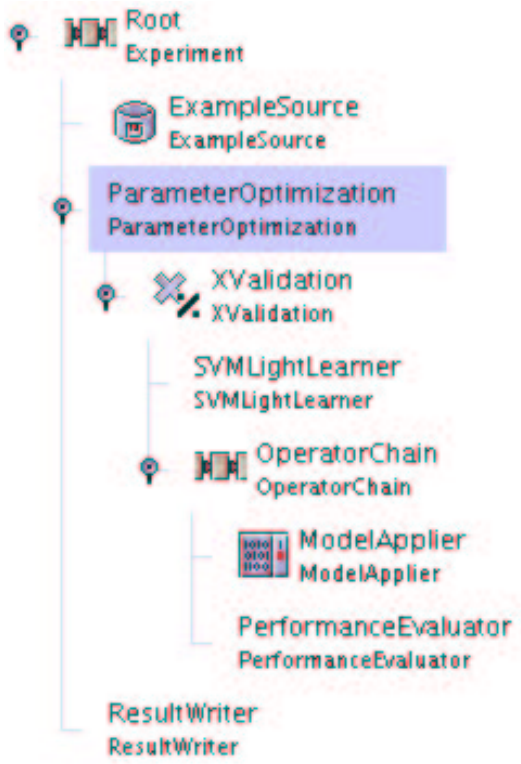


Abbildung 18.11: Parameteroptimierung unter Yale

kombiniert werden. Sobald ein Lernverfahren ein Beispiel als positiv vorhersagt, wird dieses auch auf der "Meta-Ebene" als positiv klassifiziert. Dieses Verfahren kann im allgemeinen zwei Auswirkungen auf ein Beispiel haben:

- **Fall 1:** Die positiven Beispiele, die durch die SVM als negativ klassifiziert wurden, werden bei dem zweiten Lernverfahren richtig als positiv erkannt. Die falsch-negativen Beispiele werden also durch die *ODER*-Verknüpfung zu wahr-positiven. Dies ist der ideale und wünschenswerte Fall.

- **Fall 2:** Der schlechte und zu vermeidende Fall tritt auf, wenn die negativen Beispiele, die durch die SVM richtig als negativ erkannt werden, bei dem zweiten Lernverfahren positiv klassifiziert werden. Durch die Verknüpfung auf der Meta-Ebene wird das in Wahrheit negative Beispiel positiv klassifiziert. Ein wahr-negatives Beispiel wird durch die *ODER*-Verknüpfung zu falsch-positiv.

Um den zweiten Fall möglichst zu vermeiden, ist für die Kombination mit der SVM ein Lernverfahren gewählt worden, das auf den Trainingsdaten eine hohe Precision und einen akzeptablen Recall erzielt. Bei einer 10-fachen Kreuzvalidierung erbrachte unter Yale Adaboost mit J48 als inneren Lerner diese Voraussetzung (siehe 18.5 und 18.12). Das auf diese Weise entstandene Ensemble wird in 18.13 beschrieben.

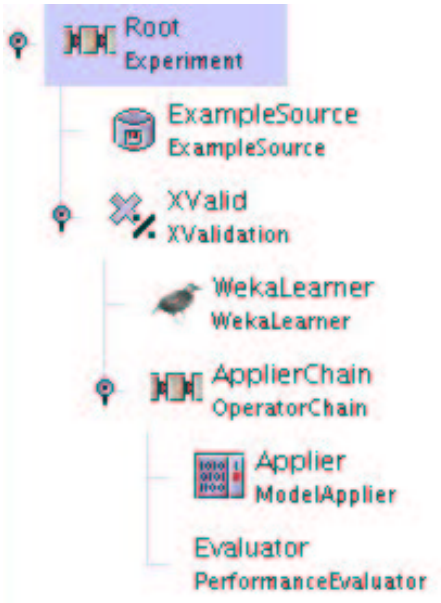


Abbildung 18.12: Adaboost mit J48 unter Yale

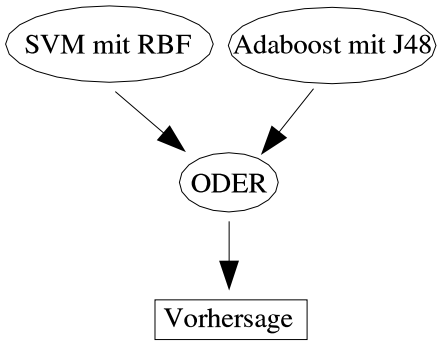


Abbildung 18.13: Funktionsweise der Kombination

Die auftretende Häufigkeit der beiden beschriebenen Fälle beschreibt den Wirkungsgrad dieses Ensembles. Bei einer genaueren Untersuchung wurde festgestellt, dass tatsächlich der erste günstige Fall im Verhältnis sechs zu eins

häufiger auftritt als der ungünstige zweite Fall. Aus diesem Grund konnte angenommen werden, dass durch eine Verknüpfung der Vorhersagen eine Verbesserung bzgl. des RMS auftritt. Nach Anwendung des Gesamtmodells auf den gesamten Trainingsdaten verbesserte sich der Wert für den RMS von 0,036 auf 0,0358.

Bayesian-Boosting

Anstatt des Adaboost mit J48 kam der Lerner Bayboost mit Y45 als inneren Lerner in frage. Allerdings konnte dieser Ansatz aus Zeitgründen nicht bearbeitet werden. Nach der Abgabe des oben beschriebenen Lösungsansatzes konnte allerdings der alternative Ansatz bearbeitet werden: Die Verknüpfung der Vorhersagen der SVM light und Bayboost mit Y45 erzielten einen RMS-Error von 0,03475, d.h. um circa eine Promille besser als die von uns abgegebene Lösung.

18.3.2 Blockweises Vorgehen für APR, RKL und TOP1

Ranking SVM

Drei der in Abschnitt 18.1.2 beschriebenen Performanzmasse (TOP1, RKL und APR) beziehen sich auf die Vorhersage einer Rangfolge der Sequenzen innerhalb eines Blocks. Der Ranking SVM Algorithmus von Thorsten Joachims [43] ist ein Lernverfahren zur Vorhersage von Rangfolgen. Hierfür muss eine Funktion aus den Trainingsbeispielen gelernt werden, die die gegebene Rangfolge der Beispiele möglichst gut approximiert. Ein statistisches Ähnlichkeitsmass für Rangfolgen ist Kendalls τ. Seien n Objekte x_i, $x \in 1, ..., n$, gemäß ihrer Rangfolge bzw. ihrer Ränge r_i sortiert, seien s_i die Ränge einer zweiten Rangfolge, mit der die erste verglichen werden soll. Dann wird q_i definiert als $q_i :=$ *Anzahl der Objekte mit* $s_j > s_i$ *und* $i \leq j \leq n$. Kendalls τ ist dann $\tau = 1 - \frac{4 \sum_{i=1}^{n} q_i}{n(n-1)}$.
Die Idee des Ranking SVM Algorithmus ist nun folgende: Es soll eine Funktion gelernt werden, die Kendalls τ maximiert. Dies wird auf lineare Funktionen (Gewichtsvektoren) beschränkt. Unter Berücksichtigung aller aus den Rangfolgen enstehenden Ungleichungen wird dieses Optimierungproblem NP-hart. Durch die Einführung einer Füllvariable kann man allerdings eine optimale Lösung approximieren und die resultierenden Gleichungen dieses Optimierungsproblems ähneln denen der Klassifikations-SVM und können daher auf die

Abbildung 18.14: YALE-Kette für das Clustering

gleiche Art gelöst werden (vergleiche Abschnitt 5.3.1).

Clustering

In der Literatur lassen sich einige Ansätze zur Verbesserung der Lernergebnisse der Ranking-SVM finden (siehe [21] und [65]). Cluster-Algorithmen sind ein Ansatz, den wir in zwei unterschiedlichen Methodiken verfolgt haben:

- ein k-Clustering Verfahren
- und ein aggregierendes hierarchisches Verfahren

Ranking-Modell mit k-Means Zuerst wurde ein k-Clustering-Verfahren mit der Absicht eingesetzt, den Datenraum in möglichst intrahomogene und interhomogene Partitionen aufzuteilen. Dazu wurde das k-Means-Verfahren (siehe Abschnitt 10.1.2) mit der Euklidischen Länge und verschiedenen k-Werten angewendet. Da die Datenmenge fast 150000 Beispiele umfasst, war es notwendig eine 10%-Stichprobe mit MiningMart zu ziehen. Auf dieser Stichprobe konnte k-means angewendet werden.

Nun wurde der Datensatz in 3 gleichgroße Beispielmengen aufgeteilt. Die erste Beispielmenge wurde gemäß der Clusterzugehörigkeit in die Cluster aufgeteilt. Auf jedem dieser Cluster wurde nun ein Ranking-Modell gelernt. Dabei ergab sich bei den Clustern ein sehr unterschiedliches Bild:

Cluster	#Beispiele	#SV's
1	5777	71
2	6770	54
3	8283	99
4	5928	136
5	5417	150
6	16055	765
7	1903	32

Tabelle 18.6: Komplexität der Cluster und der zugehörigen Ranking-Modelle

Auch die Verteilung der positiven Beispiele war in den Clustern unterschiedlich, was sich auch in der Anzahl der Stützvektoren (SV's) wiederspiegelt (siehe Tabelle 18.6). Mit höherer Anzahl von positiven Beispielen steigt die Anzahl der Ranking-Constraints und damit auch die Komplexität des Ranking-Modells.

Die 7 Ranking-Modelle wurden nun jeweils auf die zweite Beispielmenge angewendet. Dadurch erhielten wir 7 neue Ranking-Attribute. Nun wurde auf diesen 7 Attributen ein Meta-Ranking-Modell gelernt. Zur Validierung der Methode diente die dritte Beispielmenge, auf die zunächst die 7 Cluster-Ranking-Modelle angewendet wurden. Auf den Vorhersagen wurde das Meta-Ranking-SVM-Modell angewendet und nun mit der Software **PERF** bewertet. Die Ergebnisse dieser Methodik schwankten sehr stark mit der Methode wie die Beispielmengen bestimmt wurden.
Da die Bewertung der Gütemaße blockweise erfolgt, schien es sinnvoll das Stichprobenziehen der drei Beispielmengen und die Clusterzugehörigkeit ebenfalls blockweise durchzuführen, d.h. ganze Blöcke werden gesampelt. Um die Clusterzugehörigkeit eines Blockes zu bestimmen, wird eine Mehrheitsentscheidung der Clusterzugehörigkeit der Beispiele dieses Blockes vorgenommen. Die nun so erzielten Ergebnisse waren verlässlicher, da nicht mehr vom Zufall wie die Stichprobe gezogen wurde abhängig, jedoch schlechter als ein Ranking-Modell auf den Originaldaten (siehe Tabelle 18.7).

Gütemaße	Clustermethodik	Block-Clustering	Originaldaten
APR	0.44761	0.46517	0.48451
RKL	74.59477	62.52941	45.35294
TOP1	0.77778	0.84314	0.82353

Tabelle 18.7: Ergebnisvergleich der Clustermethodik (APR-Werte mit **PERF** Version 5.09[3])

Diez Clustering Ein aggregierendes, hierarchiches Clusterverfahren ist das Diez Clustering ([21]). Aufgrund seiner aggregierenden Arbeitsweise sollte es zum Einsatz kommen, um die Anzahl der eventuell auftretenden Cluster zu bestimmen.
Beim Diez Clustering bilden zunächst alle einzelnen Blöcke die initialen Cluster. Für jedes dieser Cluster wird mit der Ranking SVM ein Modell gelernt. Diese Modelle, gegeben durch die Gewichtsvektoren, werden nach ihrer Ähnlichkeit sortiert. Als Ähnlichkeitsmaß zweier Modelle wird der Kosinus verwendet, $d(\vec{w}_1, \vec{w}_2) = \frac{\vec{w}_1 \cdot \vec{w}_2}{|\vec{w}_1||\vec{w}_2|}$. Die beiden Cluster mit den ähnlichsten Modellen werden zu einem neuen Cluster vereinigt, für den ein neues Modell gelernt wird. Wenn nun die Güte der beiden einzelnen Modelle schlechter ist als die des neu erstellten Modells, wird das neue Cluster akzeptiert und der Algorithmus geht in eine neue Runde. Falls nicht, wird das Cluster verworfen und es werden aus der Ähnlichkeitsliste zwei neue Kanditaten getestet. Der Algorithmus terminiert, wenn keine neuen Cluster mehr gebildet werden können.
Die Schwierigkeit bei der Umsetzung des Diez Clustering besteht in der Auswahl eines geeigneten Gütemaßes für die Modelle. Wir verwendeten hierfür den relativen Anteil der Stützvektoren, Support Vector Rate (SVR): $SVR := \frac{Anzahl\ der\ SV}{Anzahl\ der\ Trainingsbeispiele}$. Dieses Maß ist äquivalent zum erwarteten Generalisierungsfehler der Ranking SVM (vergleiche Abschnitt 5.7.1).
Auf einer Zweiteilung der Trainingsdaten wurden auf diese Weise 7 und 9 Cluster gefunden. Bei der Anwendung des Diez Clustering auf allen Trainingsdaten wurden 15 Cluster gefunden. Somit hängt die Anzahl der zu findenden Cluster anscheinend von der Anzahl der verwendeten Blöcke ab. Das Diez Clustering sollte verwendet werden, um unabhängig von der Anzahl der Blöcke eine bestimmte Clusteranzahl zu finden, und wurde daher verworfen.

Merkmalsgenerierung

Euklidische Länge Motiviert durch die Unterschiede der Attribute (Minimum, Maximum, Durchschnitt und Median) wurde die Euklidische Länge der Beispielvektoren untersucht und als neues Attribut den Originaldaten hinzu-

gefügt. Dem Histgramm und dem Scatterplot (siehe 18.15 und 18.16) kann man entnehmen, dass diejenigen Beispiele, deren Vektor eine Euklidische Länge größer als 25000 hat, negative Beispiele sind. Das Histgramm zeigt deutlich, dass dies jedoch nur sehr wenige Beispiele sind. Wie in Abschnitt 18.3.2 gezeigt wird, gibt es bereits eine Kombination der bisherigen Merkmale, die einen sehr großen Teil der negativen Beispiele abdeckt. Daraus resultierend konnten mit diesem Attribut keine Verbesserung erzielt werden.

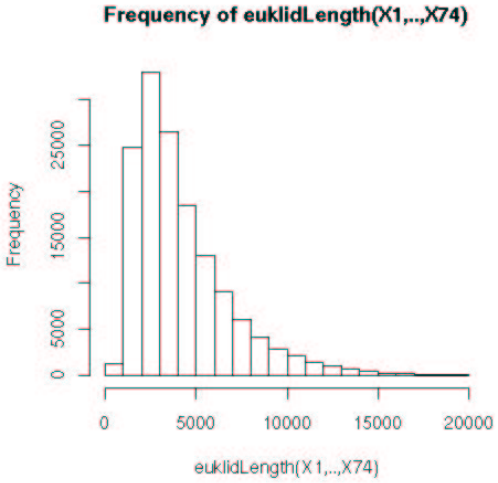


Abbildung 18.15: Histogramm der Euklischen Länge der Beispielvektoren

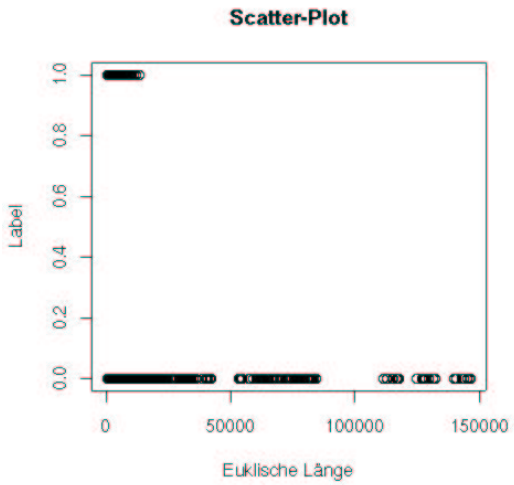


Abbildung 18.16: Korrelation der Euklidischen Länge mit dem Label

Klassifizierer Um der Ranking-SVM ein Merkmal mit hoher Korrelation mit dem Zielattribut zur Verfügung zu stellen, wurden einige Klassifizierer benutzt: J48, AdaBoost mit J48, AdaBoost mit DecisionStumps, NaiveBayes, AdaBoost mit NaiveBayes. Die Vorhersagen dieser fünf Klassifizierer wurden ebenfalls als neue Attribute den Originaldaten hinzugefügt. Gegenüber den Originaldaten brachte dies nur für APR keine merkenswerte Verbesserung.

Diskretisierung mit Entscheidungsbäumen Aufgrund der Unterschiedlichkeit der Attributausprägungen in den Blöcken wurde der Informationsgehalt der Attribute in den Blöcken untersucht. Die Beispiele eines Blockes dienten als Beispielmenge für einen Entscheidungsbaumlerner. Die Knoten dieser Entscheidungsbäume stellen Diskretisierungen dar. Aus den Knoten der Entscheidungsbäume wurden diese Diskretisierungregeln gewonnen, z.B.:

IF X >12.25 **THEN** X_{neu}=1; **ELSE** X_{neu}=0;

Dadurch entstanden 278 neue binäre Attribute, die wiederum dem Datensatz hinzugefügt wurden. Da die erwartete Verbesserung in den Performanzmaßen nicht eintrat, wurde dieser Ansatz der Merkmalskonstruktion verworfen. Dies gilt für die Original- und die normierten Daten.

BlockBoosting-Verfahren

Bisherige Bemühungen um eine Verbesserung hatten keinen großen Erfolg. Immer wieder stellte sich die Unterschiedlichkeit (der Attributausprägungen) der Blöcke als Problem dar. Insbesondere gab es Blöcke, auf denen die bisher verwendeten Lernverfahren eine schlechte Performanz hatten und viele andere auf denen eine gute Performanz in den Gütekriterien APR, RKL und TOP1 erzielt wurde. Aus diesem Grunde entstand die Idee gerade diese schlechten Blöcke gut zu lernen. Daraus entstand ein iterativer Algorithmus zur Bestimmung dieser schlechten Blöcke.[4]

Algorithmus 1 BlockBoosting

Eingabe: Blöcke B, ValidBlocks V, int *iterationen*,
int *addblocks* und Gütemaß G
TrainingsBlöcke X;
TestBlöcke T;
Model *model*;
Performanz p,p_{test};
$X \leftarrow$ wähle einige initiale Blöcke(B);
for int $iter = 1$ to iterationen **do**
  $model \leftarrow$ lerneRankingModel(X);
  $T \leftarrow B \backslash X$;
  $p_{test} \leftarrow$ getPerformance(G, T, *model*);
  $X \leftarrow X \cup$ selectWorstBlocks(T, p_{test}, *addblocks*);
end for
$p \leftarrow$ getPerformance(G, V, *model*);
Ausgabe: p;

In jeder Iteration werden solche Blöcke zum Trainieren hinzugefügt, die eine schlechte Performanz haben.

[4]Für das BlockBoosting-Verfahren wurde ein Programm in JAVA implementiert.

Die Ergebnisse mit verschiedenen Datensätzen aus einer dreifachen Kreuzvalidierung sind in Tabelle 18.8 aufgeführt.

Datensatz	Gütemaß	Iter.	addblocks	Wert
Originaldaten	APR	4	10	0.78299
Clusterattribute	APR	4	10	0.79048
Klassifizierer	APR	4	10	0.78314
Euklidische Länge	APR	4	10	0.78183
Normierung	APR	4	10	0.78763
Betrag-Normierung	APR	4	10	0.78336
Median-Normierung	APR	4	10	0.7414
Original+EBDiskretisierung	APR	4	10	0.78743
Norm+EBDiskretisierung	APR	4	10	0.77259
Block-Normierung	APR	4	10	0.80047
Originaldaten	RKL	3	10	58.30719
Clusterattribute	RKL	3	10	63.72549
Klassifizierer	RKL	3	10	58.2549
Euklidische Länge	RKL	3	10	60.22875
Normierung	RKL	3	10	43.03268
Betrag-Normierung	RKL	3	10	72.28758
Median-Normierung	RKL	3	10	124.22876
Original+EBDiskretisierung	RKL	3	10	55.86928
Norm+EBDiskretisierung	RKL	3	10	61.3137
Block-Normierung	RKL	3	10	46.22876

Tabelle 18.8: Ergebnisse des BlockBoosting-Verfahrens mit verschiedenen Datensätzen (APR-Wert mit **PERF** Version 5.11) – Fortsetzung siehe Tabelle 18.9

Mit dem BlockBoosting-Verfahren war festzustellen, dass die einzelnen Werte der Gütemaße an eine Grenze gelangt sind. Dabei wurden mit verschiedenen Datenrepräsentationen diegleichen Ergebnisse erzielt. Jedoch war keine entscheidende Verbesserung gegenüber dem Ranking-Modell aus dem Ausgangsdatensatz festzustellen.

BlockNearestNeighbor-Verfahren

Eine andere Möglichkeit die Unterschiede zwischen den Blöcken zu berücksichtigen besteht darin, dass dieser Unterschied quantifiziert wird. Ein Nearest-Neighbor–Verfahren ist in der Lage diese Unterschiede zu berücksichtigen, in-

Datensatz	Gütemaß	Iter.	addblocks	Wert
Originaldaten	TOP1	5	5	0.84967
Clusterattribute	TOP1	5	5	0.84967
Klassifizierer	TOP1	5	5	0.84967
Euklidische Länge	TOP1	5	5	0.81699
Normierung	TOP1	5	5	0.84313
Betrag-Normierung	TOP1	5	5	0.78431
Median-Normierung	TOP1	5	5	0.82353
Original+EBDiskretisierung	TOP1	5	5	0.84313
Norm+EBDiskretisierung	TOP1	5	5	0.81045
BlockNormierung	TOP1	5	5	0.83006
Beste Werte:				
Block-Normierung	APR	4	10	0.80047
Normierung	RKL	3	10	43.03268
Originaldaten	TOP1	5	5	0.84967

Tabelle 18.9: Ergebnisse des BlockBoosting-Verfahrens mit verschiedenen Datensätzen – Fortsetzung von Tabelle 18.8

dem es den Unterschied mit einem Distanzmaß quantifiziert. Um ein Nearest-Neighbor – Verfahren auf den Blöcken anwenden zu können, war es notwendig die Beispiele eines Blockes so zusammenzufassen, dass daraus ein Vektor entsteht, der diesen Block charakterisiert. Für jedes Attribut wurde über die Beispiele des Blocks Minimum, Maximum, Durchschnitt und Median bestimmt. Aus dem ursprünglichen Datensatz mit 74 Attributen entstand ein Datensatz mit 153 Beispielen (für jeden Block ein Vektor) mit 296 Attributen, die die Attributausprägungen in diesem Block beschreiben.

Abbildung 18.17 stellt das BlockNearestNeighbor-Verfahren dar[5]. Um für einen Block das Ranking vorherzusagen wurden die k-nächsten Blöcke anhand der Block2Vector-Repräsentation ermittelt. Als Distanz zwischen zwei Blöcken wurde der Euklidische Abstand zwischen den Vektoren berechnet. Die damit erhaltene Liste von Blöcken $\{B_1, ..., B_k\}$ wurde zum Lernen des Ranking-Modells genutzt. Dafür kamen jetzt aber wiederum alle Beispiele dieser ausgewählten Blöcke zum Einsatz. Das so erhaltene Ranking-Modell wurde auf den vorherzusagenden Block angewendet. Der Vorteil dieses Verfahrens liegt in der Validierung. Da jeder Block einzeln validiert werden muss, benötigt ein Leave-One-Block-Out-Vorgehen genau soviel Rechenzeit wie eine x-fache Kreuzvalidierung. Im weiteren Vorgehen haben wir uns deshalb für das Leave-One-Block-

[5]Für das BlockNearestNeighbor-Verfahren wurde ein Programm in JAVA implementiert.

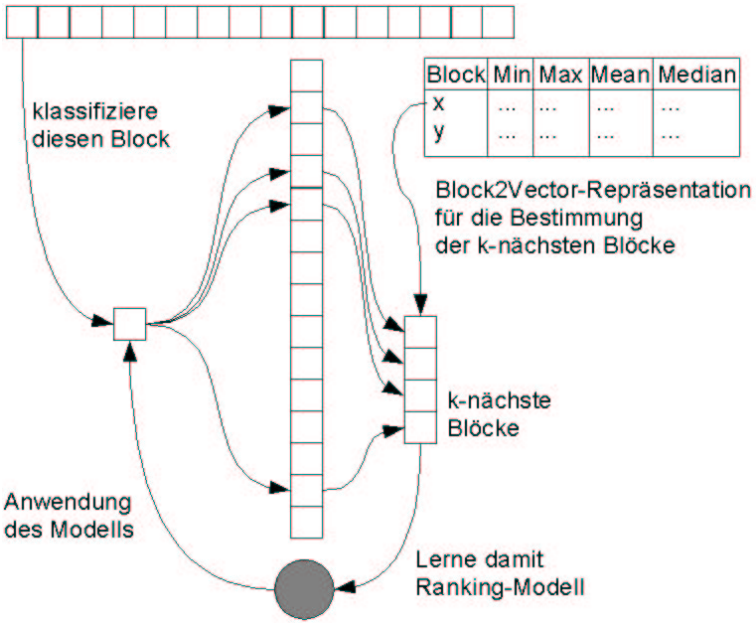


Abbildung 18.17: Das BlockNearestNeighbor-Verfahren

Out-Verfahren entschieden und erhielten damit sehr verlässliche Validierungsergebnisse. Aufgrund der Ergebnisse des BlockBoosting-Verfahrens haben wir uns bei diesem Verfahren nur auf die normierten und blocknormierten Datenrepräsentationen beschränkt. Weiterhin haben wir eine Attributauswahl für die Blockvektor-Repräsentation vorgenommen. Aus den Diskretisierungen der Entscheidungsbäume (siehe Seite 329) konnten Regeln gewonnen werden. Diese Regeln wurden nun genutzt um eine Rangfolge der Attribute hinsichtlich der Häufigkeit ihres Vorkommens in den Regeln zu bestimmen. Mit dieser Rangfolge wurde die Attributauswahl vorgenommen (siehe Tabelle 18.10). Weiterhin wurden der Parameter k – sprich die Anzahl der nächsten Blöcke – optimiert, sowie der Kostenfaktor j verbessert. Der Faktor j ist der Kostenfaktor, mit dem der Trainingsfehler auf den positiven Beispielen dem Trainingsfehler auf den negativen Beispielen überwiegt.

In Tabelle 18.11 sind die Ergebnisse nach über 100 Lernläufen zur Optimierung der Parameter zu sehen. Diese erreichten Werte stellen gegenüber dem BlockBoosting-Verfahren zumindst eine kleine Verbesserung dar, jedoch sind sie aussagekräftiger, da sie einen Leave-One-Out-Fehler bzgl. der Blöcke darstellen.

Merkmal	Häufigkeit	Merkmal	Häufigkeit	Merkmal	Häufigkeit
X49	18	X4	6	X9	2
X50	18	X5	6	X2	2
X60	14	X30	6	X15	2
X53	12	X54	6	X18	2
X1	12	X64	6	X21	2
X69	12	X46	4	X24	2
X3	12	X38	4	X34	2
X40	10	X57	4	X36	2
X55	10	X65	4	X41	2
X59	10	X70	4	X43	2
X33	8	X10	4	X48	2
X35	8	X12	4	X68	2
X44	8	X13	4	X72	2
X45	8	X23	4	X73	2
X58	8	X25	4		
X63	8	X8	2		

Tabelle 18.10: Häufigkeiten der Attribute aus den Entscheidungsbäumen der blocknormierten Daten

Datensatz	k	j	Attribute	Gütemaß	Wert
Block-Normierung	15	110	alle Attribute	APR	0.80087
Normierung	15	92	bis Häufigkeit 2	RKL	39.9085
Block-Normierung	15	110	alle Attribute	TOP1	0.8562

Tabelle 18.11: Leave-One-Block-Out-Fehler des BlockNearestNeighbor-Verfahrens

Regeln Um das BlockNearestneighbor-Verfahren zu unterstützen, wurden Regeln gesucht, die viele negative Beispiele mit hoher Genauigkeit abdecken. Dazu wurden mit MiningMart Stichproben gezogen mit unterschiedlicher Verteilung der positiven Beispiele gegenüber den negativen (siehe Abbildung 18.18). Mit Hilfe von Entscheidungsbaum- und Regellernern konnten einige Regeln gefunden werden, die mit hoher Genauigkeit negative Beispiele filtern konnten. Dabei stellte sich die Genauigkeit als ein besonderes Kriterium heraus. Deckte eine Regel nur ein paar positive Beispiele ab, so waren dies stets Beispiele, die jeweils zu einem Block gehörten, der nur ein positives Beispiel enthielt. Damit waren aber diese Regeln nicht zu gebrauchen, da nur noch negative Beispiele in diesen Blöcken übrig geblieben wären und somit auch die Gütemaße RKL und TOP1 sich dramatisch verschlechtert hätten. Jedoch gelang es

drei Regeln zu finden, die fast 60% der Daten abdeckten, dazu gehörten nur zwei positive Beispiele. Durch Kombination der Regeln mit dem Ergebnis der Ranking-SVM konnte keine Verbesserung der Lernergebnisse erzielt werden, jedoch bieten die gefundenen Regeln eine interessante Aufteilung der Daten (siehe Tabelle 18.12):

1. **IF** x51<-36.55 **AND** x9<19.75 **AND** x53<1.75 **AND** x52<2.62 **AND** x18<3.52 **AND** x19>=0.86 **AND** x2<25.65 **THEN** label=0;
2. **IF** x38<=8.32 **AND** x59<=26.5 **AND** x53<=1.59 **AND** x28<=2.55 **AND** x45 >-41 **THEN** label=0;
3. **IF** x55>-38 **AND** x8<=4.23 **AND** x38<=8.55 **AND** x53<=1.45 **AND** x21<=153.8 **AND** x63<=2.19 **AND** x23<=1.98 **AND** x70>-38 **AND** x45>-48 **THEN** label=0;

Regel	Anzahl Beispiele	davon positiv
1.	2874	1
2.	58156	1
3.	25162	0
Summe	86192	2

Tabelle 18.12: Anzahl Beispiele, die durch die jeweilige Regel abgedeckt werden

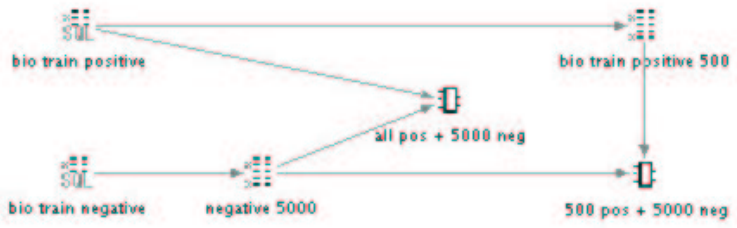


Abbildung 18.18: Stichproben in MiningMart für das Regellernen

Klassifikations-SVM mit RBF Kernel

In [56] wird als ein Anwendungsfall der SVM die Vorhersage von Proteinhomologie basierend auf Gewichten von Hidden-Markov-Modellen beschrie-

ben. In der Praxis hat sich hierfür der RBF Kernel (siehe Abschnitt 5.4.3) bewährt. Aufgrund der Ähnlichkeit zur Aufgabe des KDD Cups wurde die Klassifikations-SVM mit RBF Kernel als Lösungsansatz verfolgt.
Anstatt der Klassenzugehörigkeit wurden direkt die Funktionswerte als Ranking verwendet. Die Beträge der Funktionswerte sind proportional zum Abstand des Beispiels zur trennenden Hyperebene. D.h. bei einem positiven Beispiel mit einem hohen Abstand zur Hyperebene "ist sich die SVM sicher", dass dieses Beispiel positiv ist - analog für negative Beispiele.
In Abschnitt 18.3.1 ist die Optimierung des Parameters γ des RBF Kernels beschrieben. Zur Optimierung des Parameters j wurden zehnfache Kreuzvalidierungen mit unterschiedlichen Einstellungen für j durchgeführt. Der Parameter j ist ein Straffaktor für die Falschklassifizierung von positiven Beispielen. Die Traingsdaten wurden wie beim BlockNN Verfahren normiert. In Tabelle 18.13 sind die Ergebnisse der Kreuzvalidierungen für die Performanzmaße RKL und APR aufgelistet. Für RKL wurde $j = 200$ und für APR $j = 100$ gewählt.

Parameter j	RKL	APR
100	44,85	0,808472
200	44,34	0,803923
500	49,71	0,770857
700	47,64	0,776758
800	48,00	0,780995
1000	49,06	0,778137
1500	49,71	0,770857
2000	49,68	0,770868
2500	49,68	0,770933

Tabelle 18.13: Klassifikations-SVM mit RBF Kernel: Gemittelte Kreuzvalidierungsergebnisse bei unterschiedlichen Werten für den Parameter j.

Verbesserung durch Kombination

Um eine Verbesserung des Rankings zu erreichen, sollten die bisher besten Verfahren, das BlockNN Verfahren und die Klassifikations-SVM mit RBF Kernel, kombiniert werden. MetaLearning Ansätze brachten hierbei nicht die gewünschten Verbesserungen. So verschlechterte sich zum Beispiel das Performanzmaß RKL auf über 100 bei der Verwendung eines J48 Entscheidungsbaumes als MetaLearner. Daher wurde ein einfacheres Verfahren zur Kombination der Vorhersagen gesucht und mit der Addition der normierten Vorhersagen

gefunden.
Die Vorhersagen der beiden genannten Verfahren wurden gemäß $x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$ normiert, um eine Vergleichbarkeit der beiden vorhergesagten Rankings zu gewährleisten. Diese Rankings, gegeben durch reelle Zahlen, wurden addiert und die Summe wurde als finales Ranking verwendet. Technisch wurden die Vorhersagen, vorliegend als Textdateien, mit Hilfe eines Shell-Skriptes kombiniert, die Normierung und Addition wurde durch C-Programme durchgeführt.
Anhand der Ergebnissen aus einer zehnfache Kreuzvalidierung (siehe Tabelle 18.14) schnitt diese Art der Kombination besser ab als die einzelnen Modelle und wurde daher für die Vorhersage eines Rankings für die Performanzmaße RKL und APR verwendet.

Lernverfahren	RKL	APR
BlockNN	43,57	0,787072
Klassifikations-SVM mit RBF Kernel	44,34	0,808472
Kombination durch Addition	37,13	0,814983

Tabelle 18.14: Kombination durch Addition: Vergleich der Kombination mit Einzelmodellen, gemittelte Ergebnisse aus zehnfacher Kreuzvalidierung.

18.4 Ergebnisse des Wettbewerbs

Laut der Ergebnisse, die von der Kommission des KDD Cup 2004 am 23.07.2004 bekanntgegeben wurden, belegte die PG 445 den 9. Platz in der Gesamtwertung. Die Einzelplatzierung der PG können der nachfolgenden Tabelle entnommen werden.

Group	Platz	Top 1	Platz	RMS	Platz	RKL	Platz	APR	Gesamt
196	3.5	0.90667	8.0	0.03833	3.0	52.44667	2.0	0.84091	4.125
558	2.0	0.91333	1.0	0.03501	14.0	54.08667	1.0	0.84118	4.500
21	1.0	0.92000	5.0	0.03805	13.0	53.96000	3.0	0.83802	5.500
372	3.5	0.90667	4.0	0.03766	15.0	55.00667	6.0	0.82422	7.125
580	11.0	0.88667	2.0	0.03541	16.0	58.85333	5.0	0.82459	8.500
421	6.5	0.89333	18.0	0.03952	2.0	52.42000	10.0	0.81931	9.125
560	16.5	0.88000	6.0	0.03826	9.0	53.24000	13.0	0.81344	11.125
285	11.0	0.88667	16.0	0.03923	11.0	53.30000	8.0	0.82066	11.500
⇒555	**28.0**	**0.86667**	**14.0**	**0.03878**	**1.0**	**45.62000**	**4.0**	**0.82995**	**11.750**
587	11.0	0.88667	3.0	0.03692	24.0	64.58667	9.0	0.82006	11.750
206	5.0	0.90000	17.0	0.03941	18.0	59.11333	12.0	0.81883	13.000
591	22.0	0.87333	7.0	0.03830	4.0	52.84667	23.0	0.79938	14.000
575	22.0	0.87333	9.0	0.03838	6.5	53.06667	21.0	0.80187	14.625
584	11.0	0.88667	21.0	0.04097	23.0	61.71333	7.0	0.82420	15.500

Group	Platz	Top 1	Platz	RMS	Platz	RKL	Platz	APR	Gesamt
513	22.0	0.87333	11.0	0.03848	12.0	53.37333	19.0	0.80298	16.000
276	6.5	0.89333	22.0	0.04135	21.0	59.80667	15.0	0.80672	16.125
541	22.0	0.87333	10.0	0.03847	8.0	53.20000	27.0	0.79629	16.750
539	28.0	0.86667	12.5	0.03850	5.0	52.90000	22.0	0.79941	16.875
540	28.0	0.86667	12.5	0.03850	10.0	53.26667	30.0	0.79560	20.125
14	16.5	0.88000	26.0	0.04541	27.0	68.37333	14.0	0.80706	20.875
504	11.0	0.88667	29.5	0.05182	22.0	60.86667	25.0	0.79783	21.875
112	16.5	0.88000	19.0	0.03962	40.0	96.78667	16.0	0.80631	22.875
588	11.0	0.88667	33.0	0.05436	29.0	70.10667	20.0	0.80292	23.250
578	22.0	0.87333	24.0	0.04314	37.0	93.02667	11.0	0.81902	23.500
382	28.0	0.86667	27.0	0.04991	26.0	68.28667	18.0	0.80500	24.750
39	28.0	0.86667	25.0	0.04499	20.0	59.74000	26.0	0.79728	24.750
24	16.5	0.88000	20.0	0.03989	39.0	95.72667	29.0	0.79569	26.125
561	22.0	0.87333	15.0	0.03900	33.0	79.88667	35.0	0.77032	26.250
362	31.5	0.86000	23.0	0.04284	34.0	84.88667	17.0	0.80545	26.375
595	22.0	0.87333	32.0	0.05433	28.0	69.39333	24.0	0.79895	26.500
182	11.0	0.88667	36.0	0.09157	41.0	101.96667	31.0	0.79009	29.750
593	44.5	0.72667	29.5	0.05182	6.5	53.06667	45.5	0.64391	31.500
159	31.5	0.86000	50.0	0.16669	19.0	59.67333	28.0	0.79622	32.125
98	40.5	0.80000	31.0	0.05375	17.0	58.92667	40.0	0.73852	32.125
212	42.5	0.78667	28.0	0.05023	36.0	89.60667	41.0	0.71418	36.875
471	33.5	0.85333	38.0	0.10133	44.0	116.20667	34.0	0.77871	37.375
154	39.0	0.82000	49.0	0.15974	30.0	74.56000	32.0	0.78721	37.500
594	35.0	0.84000	54.0	0.26759	25.0	66.36667	37.0	0.76827	37.750
167	33.5	0.85333	39.0	0.10276	43.0	114.85333	38.0	0.76495	38.375
590	36.5	0.83333	48.0	0.13528	32.0	77.63333	39.0	0.76341	38.875
108	36.5	0.83333	34.0	0.05856	46.0	179.98667	42.0	0.70717	39.625
500	40.5	0.80000	58.0	8.062E4	35.0	86.94000	33.0	0.78668	41.625
3	47.0	0.64000	51.0	0.20492	31.0	75.38000	43.0	0.69035	43.000
398	38.0	0.82667	56.0	0.48079	45.0	154.10667	36.0	0.76865	43.750
407	42.5	0.78667	35.0	0.06701	56.0	557.98000	44.0	0.64550	44.375
589	44.5	0.72667	55.0	0.32017	38.0	94.35333	45.5	0.64391	45.750
544	49.5	0.52667	40.5	0.13206	47.5	364.07333	48.5	0.40547	46.500
548	49.5	0.52667	40.5	0.13206	47.5	364.07333	48.5	0.40547	46.500
545	49.5	0.52667	44.5	0.13329	49.5	375.52000	50.5	0.40466	48.500
552	49.5	0.52667	44.5	0.13329	49.5	375.52000	50.5	0.40466	48.500
26	56.0	0.07333	37.0	0.09668	55.0	416.32667	47.0	0.44952	48.750
546	52.5	0.52000	46.5	0.13337	51.5	389.48667	52.5	0.39686	50.750
554	52.5	0.52000	46.5	0.13337	51.5	389.48667	52.5	0.39686	50.750
16	54.5	0.46667	42.5	0.13241	53.5	398.24667	55.5	0.37974	51.500
581	54.5	0.46667	42.5	0.13241	53.5	398.24667	55.5	0.37974	51.500
264	57.5	0.02000	53.0	0.23782	42.0	114.82000	54.0	0.39570	51.625
187	59.0	0.01333	52.0	0.22880	57.0	810.53333	57.0	0.01727	56.250
298	57.5	0.02000	57.0	0.99076	58.0	851.46000	58.0	0.01453	57.625
476	46.0	0.68667	-	-	-	-	-	-	-

Tabelle 18.15: Platzierungen der Wettbewerbsteilnehmer (Ergebnisse der Projektgruppe sind markiert)

Kapitel 19

Konferenzteilnahme

Dirk Dach, Christophe Foussette

19.1 Einleitung

Die diesjährige KDD Konferenz der ACM SIGKDD fand vom 22.08. bis zum 25.08. in Seattle, WA, statt. Im Rahmen dieser Konferenz wurden am 22.08. die Ergebnisse des KDD-Cups vorgetragen. Die PG belegte bei der Protein-Homologie Aufgabe bezüglich des Performanzkriteriums RKL den ersten Platz. Daher wurde sie eingeladen, an der Konferenz teilzunehmen. Die PG wurde von zwei Teilnehmern und einem Betreuer in Seattle vertreten. Im Folgenden werden die Lösungen der Gesamtsieger vorgestellt.

19.2 Protein-Homologie Aufgabe

In der Gewinnerlösung für die Protein-Homologie Aufgabe wurden zuerst anhand einer 2-fachen Kreuzvalidierung mehrere Lernverfahren getestet. Anhand der Accuracy wurden anschließend drei Verfahren ausgewählt, die die besten Werte für dieses Kriterium lieferten. Hierbei handelte es sich zum einen um eine Support Vector Machine mit linearem Kernel, wobei eine logistisches Modell benutzt wurde, um die Ausgabewerte der Support Vector Machine in Wahr-

scheinlichkeiten umzuwandeln. Als zweites Verfahren wurde Adaboost mit 10 unbeschnittenen Entscheidungsbäumen eingesetzt. Beim dritten Verfahren bediente man sich des Random Forest, womit ca. 100.000 Regeln generiert wurden. Als entgültiges Modell wurde ein Ensemble der drei Verfahren verwendet. Dabei wurde die Entscheidung über die vorhergesagte Klasse auf eine ungewöhnliche Art getroffen: Wenn die Vorhersage der Support Vector Machine und der Regeln übereinstimmte wurde der Durchschnitt der Wahrscheinlichkeiten der beiden Verfahren als Vorhersage benutzt. Falls diese sich widersprachen wurde die Vorhersage von Adaboost benutzt um eine Entscheidung zu treffen. Damit wurde bei TOP1 der dritte , bei RMSE der achte, bei RKL der dritte und bei APR der 2. Platz erreicht. Das Ensemble ergab bei drei der vier Vorhersagen den besten Wert, bei RMSE allerdings wäre die Vorhersage von Adaboost alleine am besten gewesen. Erwähnenswert ist weiterhin, dass die Gewinnerlösung als einzige der Abgaben die auf den ersten zwanzig Positionen abgeschnitten haben dieselbe Vorhersage für alle Klassen abgeliefert hat. Daher belegte die Gewinnerlösung auch für keines der vier Performanzmaße den ersten Platz, trotzdem reichte es insgesamt zum Gewinn des KDD-Cups. Dies unterstützt die in der 'Organizers Presentation' vorgestellte These, dass es besonders bei der Protein-Homologie Aufgabe von Vorteil war getrennt Abgaben für die vier Disziplinen abzugeben. Aus der Gewinnerlösung und aus anderen Abgaben wurde z.B. deutlich, dass es kontraproduktiv war, gleichzeitig für den RMSE und für APR zu optimieren. Der Gewinner merkte noch an, dass es noch Raum für Verbesserungen gibt: Eine getrennte Optimierung für jedes der vier Maße, Bagging, um bessere Wahrscheinlichkeitsschätzungen zu erhalten oder weitere Datenanalyse, z.B mittels Hauptkomponentenanalyse. Die Projektgruppe PG445 schnitt besonders gut beim Performanzmaß RKL ab, wo sie mit Abstand den ersten Platz belegte und während der Konferenz mit einer 'Honorable Mention' ausgezeichnet wurde, die zwei Repräsentanten der Projektgruppe während des ersten Tages der Konferenz in Seattle von den KDD-Cup Chairs Rich Caruana und Thorsten Joachims überreicht bekamen. Der für RKL verwendete Ansatz wurde in leicht abgewandelter Form auch für APR und TOP1 verwendet. Hier wurden der 4. bzw der 28. Platz erreicht. Beim RMS wurde mit eine Kombination aus Adaboost und einer Support Vector Machine mit RBF-Kernel der 14. Platz erreicht. Betrachtet man die Gewinnerlösung so fällt auf, dass hier mit Adaboost alleine ein besseres Ergebnis erzielt wurde. Insgesamt erreichte die Projektgruppe einen guten 9. Platz, der aber bei näherer Betrachtung der Umstände und der Gewinnerlösung durchaus hätte besser ausfallen können.

19.3 Quantenphysik Aufgabe

Bei der Physik-Aufgabe kristallisierte sich eine Gruppe als eindeutiger Sieger heraus, da sie in 3 der 4 Disziplienen den ersten Platz belegte. Hier wurde eine sehr detailierte Analyse der Variablen und deren Zusammenhänge untereinander sowie den daraus resultierenden Werten der Klassenvariablen durchgeführt. Es ergaben sich 639 Vorhersageregeln, bei den z.T. bis zu 3 Variablen kombiniert wurden. Da falsche Vorhersagen von Null und Eins bei der Kreuzentropie zu sehr großen Strafen führten (eine falsche Vorhersage dieser Art führte sofort dazu, dass man keine Chance mehr auf einen der vorderen Plätze hatte) wurden Werte von 1 auf 0.995 und Werte von 0 auf 0.005 abgeändert. So wurden mehrere kleine Strafen in Kauf genommen, um ein Desaster zu verhindern. 16 Gruppen auf den hinteren Plätzen erlebten dieses Desaster. Insgesamt erreichten die Gewinner damit den 1. Platz für die Maße Cross-Entropie, Q-Score und Area under the ROC Curve. Bei Accuracy reichte es immerhin zum 2. Platz.

Teil IV

Anhang – Entwickelte Werkzeuge

Anhang A

Data2DB

Holger Flick, Daniel Hakenjos, Felix Jungermann

A.1 Beschreibung

Data2DB wurde entwickelt um kommaseparierte Textdateien, im Weiteren als Rohdaten bezeichnet, in Datenbanken einzulesen, wobei die Datentypen der einzelnen Spalten automatisiert bestimmt werden können. Zudem hat der Benutzer die Möglichkeit, das Ergebnis der Datentypanalyse vor dem Einfügen der Datensätze in die Datenbank anzupassen. Besonders flexibel ist Data2DB dadurch, daß auch eine durch den Benutzer geschriebene Parser-Klasse zur Aufbereitung der Daten eingebunden werden kann (vgl. Abschnitt A.6). Die Namend er einzelnen Attribute müssen in einer separaten Textdatei zur Verfügung gestellt werden.

Um in der Datenbank noch erkennen zu können, welche Daten nicht oder fehlerhaft spezifiziert wurden, bietet Data2DB auch eine Handhabung für NULL-Werte.

A.2 Voraussetzungen

Data2DB nutzt eine zusätzliche Klassenbibliothek, die die Erstellung eines konfigurierbaren Log-Files ermöglicht. Diese ist in den Klassenpfad einzubinden.

Die Anwendung wird bisher über die Konsole gesteuert. Weiterhin sind Konfigurationsdateien notwendig geworden, um die Eingaben auf der Kommandozeile auf ein zumutbares Maß zu verkürzen.

A.3 Konfigurationsdatei

Die Anwendung sucht nach dem Start im Unterverzeichnis `etc/` (ausgehend vom Verzeichnis, aus dem gestartet wurde) nach der Konfigurationsdatei `Data2DB.cfg`. Parameter sind in der Form `Parameter = Wert` anzugeben.

```
Gandalf:/home # export CLASSPATH=/home/pg445/share/jakarta-log4j-1.2.8/
...dist/lib/log4j-1.2.8.jar:$CLASSPATH
```

Parameter	Beschreibung	Vorgabe
`DBServer`	Adresse des Datenbankservers	`hamlet.cs.udo.edu`
`DBPort`	Port des Datenbankservers	`1521`
`DBName`	Name der Datenbank	`fbi`
`DBUser`	Benutzeraccount für Datenbankzugriff	`pg445_bus`
`DBPasswd`	Paßwort für Datenbankzugriff	
`DBTyp`	verwendetes Datenbanksystem (bisher nur Oracle)	`oracle`
`Delimitter`	Trennzeichen in der Datendatei	`,`
`ParserClass`	Name der Parser-Klasse; diese ist für jeden KDD-Cup individuell zu implementieren. Eine Beispiel-Klasse namens `MyParser` ist im JAR zu Data2DB enthalten.	`MyParser`
`rec_start`	ab diesem Datensatz werden **alle** Datensätze zur Datentypanalyse genutzt	`1000`
`rec_end`	bis zu diesem Datensatz werden **alle** Datensätze zur Datentypanalyse genutzt	`2000`
`rec_first_random`	Von den Datensätzen, die vor *rec_start* liegen, wird nur jeder *rec_first_random*-te zur Datentypbestimmung berücksichtigt	`100`
`rec_last_random`	Von den Datensätzen, die hinter *rec_end* liegen, wird nur jeder *rec_first_random*-te zur Datentypbestimmung berücksichtigt	`100`
`log4j.rootLogger`	gibt an, ab welchem Wichtigkeitsgrad auf die Konsole Fehlermeldungen geschrieben werden sollen (vgl. Abschnitt A.5.2)	`ERROR, A1`
`log4j.logger.Data2DB`	gibt an, ab welchem Wichtigkeitsgrad in die Log-Datei Fehlermeldungen geschrieben werden sollen	`INFO, A2`

Es ist anzuraten, die Parameter zur Datentypbestimmung der Datensatzanzahl der Rohdaten anzupassen, da es sich bei der Datentypbestimmung um einen zeitintensiven Vorgang handelt.

Die Konfigurationsdatei enthält weitere Parameter für die Funktionalität der Log-Datei. Diese Werte sind optimal abgeglichen und sollten nicht verändert werden.

A.4 Verwendete Dateien

Data2DB generiert und verwendet eine Vielzahl von verschiedenen Dateien, um, die Rohdaten in die Datenbank nach den Wünschen des Anwenders einfügen zu können.

- Eingabedateien -

Suffix[1]	Beschreibung
*.txt	Rohdaten (Textdatei), Datensätze zeilenweise, Attributwerte kommaseparierte
*.att	Bezeichnung der Attribute; pro Zeile ein Attribut
*.nul	Liste aller `NULL`-Werte; pro Zeile ein Wert

- Ausgabedateien -

Suffix	Beschreibung	erzeugt durch
*.rec	Typisierung der Attribute[2]	Aktion 'r'
*.log	Log-Datei	immer

A.5 Programmbeschreibung

A.5.1 Typisierung

Es wird für jedes Attribut in der `rec`-Datei der Typ und die Länge ermittelt. Diese Datei kann vor dem Einlesen in die Datenbank editiert werden. Spezielle Attribute, die Modifikationen durch den Parser (vgl. Abschnitt A.6) benötigen, sind mit '$*$' zu kennzeichnen. Alle Angaben sind durch Kommata zu trennen.

[1] es handelt sich um Vorschläge, da diese Dateien durch den Benutzer vorgegeben werden und somit beliebig benannt werden können

[2] für das Einfügen in die Datenbank ist diese Datei eine Eingabedatei

Typ	Bedeutung
C	alphanumerisch
N	numerisch, Ganzzahl
F	nummerisch, Gleitkomma
D	Datum
T	Zeit

Beispiel

1. `id, C, 20`
2. `yob,D,0,*`

Das obige Beispiel zeigt die Angabe eines 'id'-Feldes, welches alphanumerisch mit der Länge 20 klassifiziert wurde. Weiterhin wird das Feld 'yob' als Datumstyp in die Datenbank eingetragen, wobei beim Einfügen der Werte für dieses Attribut der Parser angewendet wird.

A.5.2 Log-Datei

Jede Zeile der Log-Datei enthält zuerst eine Angabe zur Wichtigkeit der Information. Weiterhin kann konfiguriert werden, welche Meldungstypen überhaupt in die Log-Datei aufgenommen werden sollen. Es gibt die Kategorien

- DEBUG (Ablaufbezogene Informationen, zur Fehlerfindung),
- INFO (Informationen),
- WARN (Warnungen),
- ERROR (Fehlermeldungen)
- und FATAL (Fehlermeldungen mit anschl. Programmabbruch).

Wird z.B. der entsprechende Parameter in der Konfigurationsdatei auf 'WARN' gesetzt, so werden nur Warnungen, Fehler und Fehlermeldungen, die zum Abbruch führen, ausgegeben.

A.5.3 Programmstart

```
Gandalf:/home/Data2DB # java Data2DB <Rohdaten> < r|t|i >
```

- **Rohdaten:** Dateiname mit vollständigem Pfad zur Textdatei mit den Rohdaten
- **Aktionsfolge:** gibt an, welche Aktionen durchgeführt werden sollen

A.5.4 Aktionsfolge

Die folgende Tabelle listet alle Aktionskürzel auf und welche Programmfunktionen dadurch aufgerufen werden:

r	Analysiert die Datentypen der Rohdaten, wobei nur eine Stichprobe gemäß der in der Konfigurationsdatei gesetzten Parameter betrachtet wird.
t	Erstellt eine Tabelle gemäß den in der rec-Datei angegebenen Datentypen und den in der att-Datei spezifizierten Attributnamen. Der Anwender wird zur Eingabe des Namens für die Tabelle aufgefordert..
i	Fügt die Datensätze in die Tabelle ein. Der Anwender wird zur Eingabe des Namens für die Tabelle aufgefordert.

Die Aktionen können miteinander kombiniert werden, wie es die folgenden Beispiele erkläutern.

Beispiele

```
1. Gandalf:/home/Data2DB # java Data2DB kdd98.txt rti
2. Gandalf:/home/Data2DB # java Data2DB kdd98.txt ti
3. Gandalf:/home/Data2DB # java Data2DB kdd98.txt i
```

1. Typisiert die Daten der Datei `kdd98.txt`, erstellt rec-Datei, erstellt Tabelle in der Datenbank und fügt die Datensätze ein. Diese Anweisung ist nur dann sinnvoll, wenn man sicher ist, daß die automatische Erkennung die korrekten Datentypen liefert und keine Aufbereitung der Daten stattfinden muß.

2. Erstellt Tabelle in der Datenbank und fügt die Datensätze ein. Dies ist der normale Arbeitsschritt, der nach Überarbeitung der rec-Datei ausgeführt wird.

3. Fügt Datensätze in Tabelle ein; die rec- und att-Dateien müssen bereits vorliegen.

A.6 Datenaufbereitung über den Parser

Bei `Parser` handelt es sich um ein vordefiniertes Interface, welches durch den Benutzer konkret implementiert werden kann. Über den Parser können z.B. Werte wie '9902' in Datumswerte umgeformt und bei der Datensatzerfassung direkt im gewünschten Datumsformat in die Datenbank eingefügt werden. Die zu benutzende Parser-Klasse ist in der Konfigurationsdatei einzutragen und im Klassenpfad zur Verfügung zu stellen.

Kern des Parsers ist die Methode 'change. Über diese Methode hat man die Möglichkeit für jedes Attribut, die jeweilige Bezeichnung wird in 'attribut' übergeben, eine Datenumformung durchzuführen. Weitere Parameter sind der Typ des Attributes ('type') und der umzuformende Wert ('inputValue'). Die Methode muß den neuen Attributwert erhalten und zwar in der Form, wie er in eine SQL Insert-Anweisung einzutragen wäre.

A.6.1 Beispiel

Gegeben sei das Attibut 'DOB' in der Form 'JJMM'. Ein möglicher Attributwert wäre also '9902', wenn eine Person im Februar 2002 geboren wurde. Eine Umformung in einen Oracle-Datumstyp ist erwünscht.

Die Methode 'change' muß somit für das Attrribut 'DOB' eine Umformung durchführen. Eine vollständige Implementierung des für den KDD Cup 98

verwendeten Parsers soll hier beispielhaft für eine mögliche Implementierung dienen.

```
/*
 * MyParser.java
 * Created on 17. November 2003
 */

 /**
 * @author  Siehyun Strobel
 */

 import java.io.*;

 public class Cup98Parser implements Parser
 {

     /** Creates a new instance of MyParser */
     public Cup98Parser() {
     }
     public boolean check(String type, String inputValue) {
         //diese Methode wird nicht vom Parser aufgerufen
         return true;
     }

         private String changeDate (String wert) {

                 int z;
                 String monat;
                 String returnWert;

                 //Datum kann drei oder vierstellig sein:981 oder 9801 für januar 1998
                 if ((wert.length()<3) || (wert.length()>4))
                             return ("null");

                 //überprüfen des jahres -> ersten beiden stellen
                 z=Integer.parseInt(wert.substring (0,2));
                 if (z>98) {
                             return ("null");
                 }

                 /*Datum hat im Kddcup 98 immer das Format YYMM
                   prüfe, ob der Monat zwischen 1 und 12 liegt
                   Monat kann ein- oder zweistellig sein*/

                 z=Integer.parseInt(wert.substring (2));
                 if ((z<1) || (z>12)) {
                             return ("null");
                 }

                 if (wert.length()==3)
                 //eine 0 einfügen, wenn Datum der Form 981 ist
                             monat = "0"+wert.substring(2);
                 else
                         monat = wert.substring(2);

                 returnWert=wert.substring(0,2)+monat;

                 return ("to_date(\'"+returnWert+"\',\'YYMM\')");
         }

             private String changeZip (String wert) {
                 return (wert.substring (0,5));
```

```
        }

            private String changeAge (String wert) {
                if (Integer.parseInt(wert)<=0)
                //wert.equalsIgnoreCase("0"))
                    return ("null");
                else
                return (wert);
        }


            public String change(String attribut, String type, String inputValue) {
                //nur diese Methode wird vom Parser aufgerufen!
                //System.out.println(type.startsWith("D"));
                if  (type.startsWith("D")){
                        //dies ist ein Datumswert der geändert werden muss
                        return changeDate(inputValue);
                }
                if (attribut.equalsIgnoreCase("ZIP"))
                           return(changeZip(inputValue));
                       if (attribut.equalsIgnoreCase ("age") && type.equalsIgnoreCase("I"))
                           return(changeAge(inputValue));
                return inputValue;
            }
}
}
```

A.7 Shellskript zur Datenerkennung und zur Erzeugung von Yale Attributdateien

Christophe Foussette

A.7.1 Einleitung

Data2DB berücksichtigt bei der Datenerkennung nur die ersten 10000 Zeilen des Datensatzes. Dies ist beim Datensatz des KDD Cup 2000 problematisch, da in diesem Datensatz bei vielen Attributen erst in den letzten Zeilen Werte ungleich null sind. Aufgrund der Nutzung des rechenintensiven StringTokenizers unter JAVA ist eine Ausdehnung der Datenerkennung auf alle Zeilen der Datei nicht praktikabel. Daher wurde zur Datenerkennung das Shellskript data_rec.sh geschrieben, das eine rec Datei zur weiteren Verabeitung mit Data2DB erzeugt. Eine weitere Funktionalität des Skripts ist die Erzeugung von Attributdateien für Yale in XML.

A.7.2 Funktionsweise des Skripts

Das Skript arbeitet auf Textdateien und nutzt die unter UNIX zur Verfügung stehenden Texttools. Die Datenerkennung trennt die Datei mit Hilfe von "cut" in ihre einzelnen Spalten auf. Diese einzelnen Spalten werden mit "grep" auf Zeichen untersucht, die einen Rückschluss auf den vorhanden Datentyp zulassen. So wird zum Beispiel beim Auftreten von Buchstaben auf den Datentyp char geschlossen. Die maximale Länge einer Spalte wird mit "wc" ermittelt. Beim Erzeugen der Yale Attributdatei wird hauptsächlich "sed" eingesetzt.

Hinweise zur Datenerkennung

Das Skript verwendet als Trennzeichen das Komma, und Datumsangaben werden nur im amerikanischen Format (YYYY-MM-DD) erkannt. Wenn die gegebenen Daten diesen Voraussetzungen nicht genügen, müssen die regulären Ausdrücke im Skript entsprechend angepasst werden.

A.7.3 Bedienung des Skripts

Das Skript data_rec.sh erwartet eine Option und eventuell mehrere Parameter für diese Option. Wenn data_rec.sh ohne Optionen augerufen wird, erscheint eine Fehlermeldung mit einer kurzen Erklärung, wie das Programm korrekt gestartet wird. Die beiden Funktionalitäten des Skripts werden in den nachfolgenden Abschnitten einzeln beschrieben.

Datenerkennung

Um mit data_rec.sh eine Datenerkennung durchzuführen, muss es folgendermassen aufgerufen werden:

```
Gandalf:/home # data_rec.sh -data2rec DATAFILE NAMESFILE >RECFILE
```

Hierbei ist DATAFILE die Datei, die die zu erkennenden Daten enthält, NAMESFILE ist eine Datei, in der in einzelnen Zeilen die Attributnamen gespei-

chert sind, und RECFILE ist eine rec Datei, wie sie zur weiteren Verarbeitung mit Data2DB benötigt wird.

Erzeugen einer Yale Attributdatei

Das Erzeugen einer Yale Attributdatei geschieht mit folgendem Aufruf des Skripts:

```
Gandalf:/home # data_rec.sh -rec2yale RECFILE DATASOURCE >YALEFILE
```

RECFILE ist die Datei, die durch die Datenerkennung erzeugt wurde, DATASOURCE ist der Pfadname für "default_source" in der Attributbeschreibung und YALEFILE ist die gewünschte Attributbeschreibung für Yale in XML.

Anhang B

file2sample

Christophe Foussette

B.1 Einleitung

In der Praxis sind die beim KDD Prozess auftretenden Daten meist zeilenweise in Textdateien (Flatfiles) gespeichert. Das Programm file2sample ermöglicht es, aus einem Flatfile eine Stichprobe zu ziehen oder das Flatfile in zwei disjunkte Dateien aufzuteilen.

B.2 Funktionsweise

Sowohl die Ziehung der Stichprobe als auch die Aufteilung in zwei disjunkte Dateien wird mit Hilfe gleichverteilter Zufallsvariablen durchgeführt. So ist gewährleistet, dass die Verteilung der Stichprobe der Verteilung der ursprünglichen Daten aus dem Flatfile genügt, d.h. im Sinne der Statistik: Die zu Grunde liegenden Verteilungen sind identisch.

B.3 Bedienung

B.3.1 Stichprobe ziehen

Um mit file2sample eine Stichprobe der Größe samplesize aus der Datei infile zu ziehen, und sie in der Datei outfile zu speichern, genügt der folgende Aufruf:

```
eisen:/$ file2sample infile samplesize >outfile
```

B.3.2 Daten aufteilen

Das Aufteilen der Daten in zwei dijunkte Mengen funktioniert folgendermassen. Das Flatfile infile mit x Zeilen wird in zwei Dateien, outfile1 mit *samplesize* Zeilen und outfile2 mit $x - samplesize$ Zeilen, aufgeteilt. Dies geschieht durch folgendes Kommando (es sind zwei "-" vor dem split):

```
eisen:/$ file2sample -split samplesize infile outfile1 outfile2
```

B.4 Quellcode

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define BUFFERSIZE 8192

main(int argc, char *argv[]) {
        FILE *infile,*outfile1,*outfile2;
        char *buffer;
        long lines=0,samplesize,i,*samples,r;
        int b;

        if(argc != 3 && argc != 6) {
                printf("ERROR: wrong number of parameters\n");
                printf("USAGE:\n%s {infile} {sample size}\n",argv[0]);
                printf("%s -split {sample size} {infile} {outfile1} {outfile2}\n",argv[0]);
                exit(5);
        }
```

```
if(argc == 6 && strcmp(argv[1],"-split")) {
        printf("ERROR: unknown parameter %s\n",argv[1]);
        printf("USAGE:\n%s {infile} {sample size}\n",argv[0]);
        printf("%s -split {sample size} {infile} {outfile1} {outfile2}\n",argv[0]);
        exit(5);
}

if(argc==3)
        infile = fopen(argv[1],"r");
else
        infile = fopen(argv[3],"r");

if(!infile) {
        printf("ERROR: could not open file %s\n",argv[1]);
        exit(5);
}

samplesize = atoi(argv[2]);

buffer = (char *)malloc(BUFFERSIZE);
while(fgets(buffer,BUFFERSIZE,infile))
        lines++;
fclose(infile);

if(lines < samplesize) {
        printf("ERROR: sample size exceeds infile size\n");
        exit(5);
}

srand(time(NULL));

samples = (long *)malloc(sizeof(long)*lines);
for(i=0;i<lines;i++)
        samples[i]=0;

for(i=0;i<samplesize;i++) {
        b = 1;
        while(b) {
                r = (int)((float)lines*rand()/(RAND_MAX+1.0));
                if(samples[r] == 0) {
                        samples[r]=1;
                        b = 0;
                }
        }
}

i=1;
// simple sample output
if(argc == 3) {
        infile = fopen(argv[1],"r");
        while(fgets(buffer,BUFFERSIZE,infile)) {
                if(samples[i] == 1)
                        printf("%s",buffer);
                i++;
        }
}

// split mode creates two output files
if(argc == 6) {
        infile = fopen(argv[3],"r");
        outfile1 = fopen(argv[4],"w");
        outfile2 = fopen(argv[5],"w");
        while(fgets(buffer,BUFFERSIZE,infile)) {
                if(samples[i] == 1)
                        fprintf(outfile1,"%s",buffer);
```

```
                        else
                                fprintf(outfile2,"%s",buffer);
                        i++;
                }
                fclose(outfile1);
                fclose(outfile2);
        }

        fclose(infile);
        free(buffer);
        free(samples);
}
```

Anhang C

Hilfsprogramm für den KDD-Cup 1998

Daniel Hakenjos

Für die Analysekette in YALE wurde ein Hilfsprogramm zur Bestimmung der Güte in JAVA implementiert!

Dazu wurde die Klasse "guete" implementiert. Dabei wird die Ausgabedatei <file> in der Analysekette von YALE als Eingabe benutzt. Der Anruf des Programms erfolgt widerum in YALE mit dem "CommandLineOperator".

```
Gandalf:/home/Data2DB # java guete <file>
```

Zur Bestimmung des Gütekriteriums wird folgende Summe berechnet. In der ersten Spalte der Datei steht der tatsächliche Spendenbetrag, in der zweiten Spalte die vorhergesagte Spendenhöhe und in der vierten Spalte die Spendenwahrscheinlichkeit. Ist das Produkt von Spendenwahrscheinlichkeit und vorhergesagter Spendenhöhe größer als 0.68 $, dann wird der jeweilige Spendenkandidat als Spender klassifiziert (c=1) ansonsten nicht (c=0). Danach wird die Klassifikation mit der tatsächlichen Spendenhöhe multipliziert. Von diesem Wert wird nun noch das Produkt aus Klassifikation und 0.68$ (Vertriebskosten) abgezogen. Die Summe dieses Wertes über alle Spender ergibt die gewünschte Güte.

```
import java.io.*;
import java.util.*;

public class guete{

        public String file;

        public guete(String file){
                this.file=file;
        }

        private String getNext(String next){
                if (next.equals("?")){
                        return "0.0f";
                        }
                        else return next;
        }

        public float getGuete(){
        try{
                LineNumberReader input=new LineNumberReader(new FileReader(this.file));
                int lines=0;
                float gesamt=0;
                int p=0,a=0,p_a=0;
                String zeile=input.readLine();
                String Prob,Amount,p_Amount;
                while (zeile!=null){
                        lines++;
                        StringTokenizer st=new StringTokenizer(zeile," ");
                        Amount=this.getNext(st.nextToken());
                        float amount=Float.parseFloat(Amount);
                        p_Amount=this.getNext(st.nextToken());
                        float p_amount=Float.parseFloat(p_Amount);

                        st.nextToken();

                        Prob=this.getNext(st.nextToken());
                        float prob=Float.parseFloat(Prob);
                        if (prob<0) prob=0.1f;

                        zeile=input.readLine();

                        if (prob*p_amount > 0.68 ){
                                gesamt+=amount;
                                gesamt-=0.68;
                        }
                }

                return gesamt;

                }catch(Exception e){ System.out.println(e.getMessage());}
                return -1.0f;
        }

        public static void main(String[] args){
           guete g=new guete(args[0]);
             System.out.println("Wir haben unglaubliche "+g.getGuete()+"$ eingenommen!");

        }

}
```

Anhang D

YALE-Operatoren

D.1 Die Iterating Operator Chain

Marcel Gaspar

Die Gruppe hat einen Yale-Operator auf Basis der OperatorChain entwickelt. Ziel war es, einen Yale-Operator zu haben der gewisse Schritte, wie z.B. das Lernen von 10 verschiedenen Modellen auf ein und demselben Sample selbständig iterieren kann. Die "IteratingOperatorChain" ist das Ergebnis dieser Bemühungen, sie führt ihren Kind-Operator beliebig oft durch. Wählt man als Kind eine einfache OperatorChain werden alle an diese angehängten Operatoren mehrfach iteriert, so dass also ganze Teil-Ketten mehrfach iteriert werden können. Hierbei kann dann für Datei- bzw. Modell-Namen die geladen oder gespeichert werden sollen ein "%a" an den Namen angehängt oder eingefügt werden. Das "%a" wird dann während der Ausführung durch die Nummer des aktuellen Iterationsschrittes ersetzt. Ein Beispiel: Erhält die IterationgOperatorChain als Startparameter 1, als Endparamter die 5 und als Kind-Operator einen "ModelLoader" mit Paramter "Beispiel%a.model" so werden die Modelle "Beispiel1.model" bis "Beispiel5.model" geladen.

D.1.1 Quelltext

```
package pg445;

import edu.udo.cs.yale.operator.*;
import java.util.List;
import java.util.ListIterator;
import edu.udo.cs.yale.operator.parameter.ParameterTypeString;
import edu.udo.cs.yale.operator.parameter.ParameterTypeInt;



public class IteratingOperatorChain extends OperatorChain {

    private List operators;

    public IteratingOperatorChain() {
super();
    }

    public List getParameterTypes() {
List types = super.getParameterTypes();
types.add(new ParameterTypeInt("start", "integer to start iteration at (default: 1)", 0, Integer.MAX_VALUE, 1));
types.add(new ParameterTypeInt("end", "integer to end iteration at", 0, Integer.MAX_VALUE, 10));
return types;
    }


    public Class[] checkIO(Class[] input) throws IllegalInputException {
for (int i = 0; i < getNumberOfOperators(); i++) {
    Operator o = getOperator(i);
    input = o.checkIO(input);
}
return input;
    }


    /** Returns the maximum number of innner operators. */
    public int getMaxNumberOfInnerOperators() { return 1; }

    /** Returns the minimum number of innner operators. */
    public int getMinNumberOfInnerOperators() { return 1; }


    public IOObject[] apply() throws OperatorException {
int start = getParameterAsInt("start");
int end = getParameterAsInt("end");

for (int k = start; k<end; k++) {
super.apply();
}

return super.apply();
    }

    public int getNumberOfSteps() {
return getNumberOfChildrensSteps() + 1;
    }

    public Class[] getInputClasses() {
return new Class[0];
    }

    public Class[] getOutputClasses() {
return new Class[0];
    }

}
```

D.2 MajorityVoter

Dirk Dach

D.2.1 Einleitung

Der Majority Voter ist ein neuer Yale-Operator, der zum Meta-Lernen verwendet werden kann. Er erhält als Eingabe mehrere Vorhersagen unterschiedlicher Modelle und erzeugt daraus eine neue Vorhersage. Wie der Name des Operators schon vermuten lässt, ist diese Vorhersage die Mehrheit der Vorhersagen der einzelnen Modelle.

D.2.2 Funktionsweise

Der Operator bekommt als Eingabe ein Example Set. Dessen Attribute müssen nominal sein und den gleichen Wertebereich haben (z.B true, false). Der MajorityVoter zählt für jedes Beispiel die Häufigkeit der verschiedenen Labels und wählt als Vorhersage das am häufigsten vorkommende. Bei gleichen Häufigkeiten wird zufällig ein Label gewählt. Falls im Beispiel Werte fehlen werden nur die vorkommenden Labels gezählt. Fehlen alle Werte wird zufällig eine Vorhersage getroffen.

D.2.3 Bedienung

Die Einbindung in Yale erfolgt als Plugin, d.h es muss ein .jar-File erstellt werden und ins Verzeichnis YALEHOME/lib/plugins kopiert werden. Der Operator kann dann in die Analysekette eingefügt werden. Weitere Parametereinstellungen sind nicht nötig. Der Quellcode sowie die kompilierte Version befinden sich im Infolayer unter dem Punkt Tools.

D.2.4 Quellcode

```
package pg445;

import edu.udo.cs.yale.operator.Operator;
import edu.udo.cs.yale.operator.OperatorException;
import edu.udo.cs.yale.operator.IOObject;
import edu.udo.cs.yale.example.ExampleSet;
import edu.udo.cs.yale.example.Example;
import edu.udo.cs.yale.example.ExampleReader;
import edu.udo.cs.yale.example.Attribute;
import edu.udo.cs.yale.example.DataRow;
import edu.udo.cs.yale.example.DataRowReader;
import edu.udo.cs.yale.tools.LogService;

import java.util.*;

/**
 * This operator generates a predicted label and sets it to the majority of predictions made in each row.
 * It needs nominal attributes that all have the same possible values (e.g. true, false or yes, no).
 */
public class MajorityVoter extends Operator {

    private static final Class[] INPUT_CLASSES = { ExampleSet.class };
    private static final Class[] OUTPUT_CLASSES = { ExampleSet.class };

    public IOObject[] apply() throws OperatorException {
ExampleSet eSet = (ExampleSet)getInput(ExampleSet.class, false);
String[] labels = null;
Attribute[] attributes = eSet.getExampleTable().getAttributes();
Random generator = new Random();

// Predicted Label erzeugen
Attribute predictedLabel = new Attribute((Attribute)eSet.getLabel().clone(), ExampleSet.PREDICTION_NAME);
   eSet.getExampleTable().addAttribute(predictedLabel);
   eSet.setPredictedLabel(predictedLabel);

boolean ok = true;
String message = null;

if (attributes!=null) {
labels = new String[attributes[0].getNumberOfClasses()];
Collection c1 = attributes[0].getValuesAsString();
Iterator it1 = c1.iterator();
for (int j=0;j<labels.length;j++)
labels[j]=(String)it1.next();
}
else {
message = "no attributes";
ok = false;
}

for (int i=1;(i<attributes.length) && ok; i++) {

Collection c2 = attributes[i].getValuesAsString();
Iterator it2 = c2.iterator();

if (c2.size() != labels.length) {
message = "different number of classes in attributes";
ok = false;
}
else {
int k = 0;
while(it2.hasNext()) {
if (!(it2.next().equals(labels[k]))) {
message = "different classnames in attributes";
ok = false;
}
k++;
}
}
}

if (ok) {
LogService.logMessage("performing Majority Vote...", LogService.MINIMUM);
int[] counter = new int[labels.length];
double[] values = new double[labels.length];
DataRowReader r = eSet.getExampleTable().getDataReader();

double valueGenerator = 1.0;
for (int i=0; i<values.length; i++) {
values[i] = valueGenerator;
valueGenerator++;
}
```

```
// Alle Zeilen durchgehen
int zeilennummer=0;
while (r.hasNext()) {
DataRow row = r.next();

// Zählen der Labels
for (int i=0; i<attributes.length; i++) {
for (int j=0; j<values.length; j++) {
if (row.get(attributes[i]) == values[j] &&
    !(attributes[i].getName().equals(eSet.getLabel().getName()))) {
counter[j]++;
}
}
}
// Index des Maximum bestimmen
int maxIndex = 0;
boolean allZero=true;
boolean remis=false;

for (int i=0; i<counter.length; i++) {
if (counter[i] >= counter[maxIndex]) {
maxIndex=i;
allZero=false;
}
}
if ((counter[(counter.length)-1]==0) && (maxIndex==(counter.length-1)))
allZero=true;
if (counter[0] == counter[1])
remis=true;

// counter zurücksetzen
for (int i=0;i<counter.length;i++)
counter[i]=0;

// predicted Label schreiben
if (allZero) {
double zufall = generator.nextDouble();
zufall = zufall*labels.length;
int vote=0;
for (int i =0;i<labels.length; i++) {
if ((zufall>=i) && (zufall<i+1))
vote = i+1;
}
row.set(eSet.getPredictedLabel(), vote);
}
else if (remis) {
double zufall = generator.nextDouble();
zufall=zufall*2;
if (zufall<=1)
row.set(eSet.getPredictedLabel(),values[0]);
else
row.set(eSet.getPredictedLabel(),values[1]);

}
else {
row.set(eSet.getPredictedLabel(),values[maxIndex]);
}
}

}
else
System.out.println("Majority Vote failed: "+message);

return new IOObject[0];
    }


    public Class[] getInputClasses() { return INPUT_CLASSES; }
    public Class[] getOutputClasses() { return OUTPUT_CLASSES; }
}
```

D.3 Perforator

Dirk Dach, Felix Jungermann

D.3.1 Einleitung

Da beim diesjährigen KDD-Cup eine Vielzahl verschiedener Bewertungsmaße relevant waren, wurde vom KDD-Cup-Kommitee ein Werkzeug zur Überprüfung der Bewertungsmaße bereitgestellt.
Bei dem Werkzeug handelt es sich um ein C-Programm, das nur von der Kommandozeile aufgerufen werden kann.
Aus diesem Grund entschieden wir uns dafür, einen Yale-Operator zu entwickeln, um das „offizielle“ KDD-Tool in unsere Yale-Ketten einzubinden.

D.3.2 Importieren des Perforators

Um den Perforator in Yale zu benutzen, muss man die Datei pg445.jar in das Verzeichnis *YALE-HOME/lib/plugins* kopieren. Unter Umständen muss das Verzeichnis *plugins* erst angelegt werden.

D.3.3 Benutzen des Perforator innerhalb von Yale

Möchte man den Perforator in eine Yale-Kette einbauen, so wählt man aus der Gruppe *pg445* den Operator *Perforator* aus.
Wählt man den nun in der Yale-Kette vorhandenen Operator aus, so kann man eine Vielzahl von Optionen auswählen.
Für den KDD-Cup waren vor allem *Accuracy, Root Mean Squared Error, Mean Cross-Entropy, ROC area, Mean Average Precision, Top 1, Rank of *last* positive case* und *Slac Q-Score* wichtig, da diese die Bewertungsmaße sind, die für den KDD-Cup entscheidend sind.
Da sich das Format der Abgabe zwischen Biologie- und Physik-Datensatz unterscheidet, muss man, sofern man Biologie-Daten bearbeitet, noch das Bio-Feld auswählen.

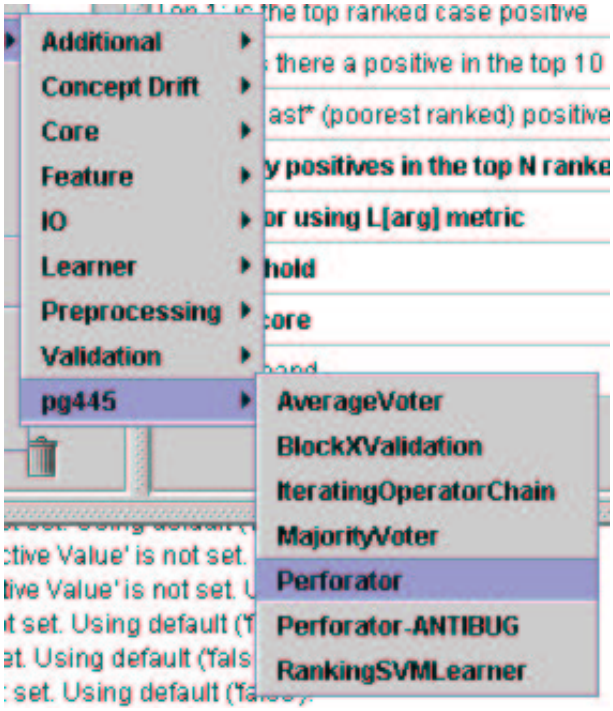


Abbildung D.1: Auswahl des Perforators in Yale

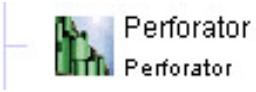


Abbildung D.2: Der Perforator innerhalb einer Yale-Kette

Sofern die Daten konform zum „offiziellen“ Tool sind, erhält man eine zufriedenstellende Ausgabe.

D.3.4 Quelltext des Perforator

```
/*
 *  YALE - Yet Another Learning Environment
 *  Copyright (C) 2001, 2002, 2003
 *      Simon Fischer, Ralf Klinkenberg, Ingo Mierswa,
 *          Katharina Morik, Oliver Ritthoff
 *      Artificial Intelligence Unit
 *      Computer Science Department
 *      University of Dortmund
 *      44221 Dortmund,  Germany
 *  email: yale@ls8.cs.uni-dortmund.de
 *  web:   http://yale.cs.uni-dortmund.de/
 *
 *  This program is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU General Public License as
 *  published by the Free Software Foundation; either version 2 of the
 *  License, or (at your option) any later version.
```

Key	Value
Bio	☐
Accuracy	☐
Root Mean Squared Error	☐
Mean Cross-Entropy	☐
ROC area	☐
ROC area up to 50 negative examples	☐
Sensitivity	☐
Specificity	☐
Negative Predictive Value	☐
Positive Predictive Value	☐
Precision	☐
Recall	☐
F1 score	☐
Precision/Recall Break Even Point	☐
Mean Average Precision	☐
Lift (at threshold)	☐
Top 1: is the top ranked case positive	☐
Top 10: is there a positive in the top 10 ranked cases	☐
Rank of *last* (poorest ranked) positive case	☐
How many positives in the top N ranked cases	
Norm error using L[arg] metric	
Set threshold	
Slac Q-score	
perfcommand	/home/pg445/share/tools/perf_newversion/perf

Abbildung D.3: Die Perforator-Optionen

```
 *
 *  This program is distributed in the hope that it will be useful, but
 *  WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 *  General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
 *  USA.
 */
package pg445;

import edu.udo.cs.yale.operator.parameter.*;
import edu.udo.cs.yale.operator.OperatorException;
import edu.udo.cs.yale.operator.*;
import edu.udo.cs.yale.example.*;
import java.io.FileNotFoundException;
import edu.udo.cs.yale.tools.LogService;
import edu.udo.cs.yale.tools.ParameterService;
import edu.udo.cs.yale.operator.performance.*;
import edu.udo.cs.yale.operator.parameter.*;

import java.util.*;
import java.io.*;
import java.lang.*;

/** A performance evaluator is an operator that expects a test {@link ExampleSet} as
 *  input, whose elements have both true and predicted labels, and delivers as output a
 *  list of performance values according to a list of performance criteria that
 *  it calculates.
 *  <br/>
 *  All of the performance criteria can be switched on using boolean parameters. Their
 *  values can be queried by a {@link edu.udo.cs.yale.operator.ExperimentLogOperator} using
 *  the same names. The main criterion is used for comparisons
 *  and need to be specified only for experiments where performance vectores are compared,
 *  e.g. feature selection experiments.
 *  <br/>
 *  Additional user-defined implementations of {@link PerformanceCriterion} can be specified
 *  by using the parameter list <var>additional_performance_criteria</var>. Each key/value pair
 *  in this list must specify a fully qualified classname (as the key), and the a string parameter
 *  (as the value) that is passed to the constructor. Please make sure that the class files are in
```

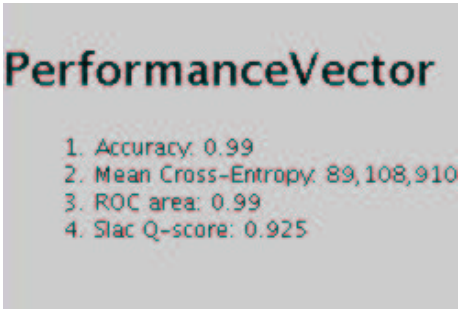


Abbildung D.4: Ausgabe des Perforators in Yale

```
 *  the classpath (this is the case if the implementations are supplied by a plugin) and that they
 *  implement a one-argument constructor taking a string parameter.
 *  <br/>
 *  Other implementations than the simple comparator, that only compares the main criterion,
 *  can be specified using the parameter <var>comparator_class</var>. This may for instance
 *  be useful if you want to compare performance vectors according to the weighted sum of the
 *  individual criteria. In order to implement your own comparator, simply subclass
 *  {@link PerformanceComparator}.
 *
 *  @yale.xmlclass PerformanceEvaluator
 *  @author Simon
 *  @version $Id: Perforator_final.tex,v 1.5 2004/09/30 10:00:15 foussett Exp $
 */
public class Perforator extends Operator {

    private static final String[] commands=new String[]{
     "/home/pg445/share/perf.linux/perf",
"/home/pg445/share/tools/perf_experimental/perf.experimental",
"/home/pg445/share/tools/perf_newversion/perf"
};


    private static final String[] CRITERIA_NAMES = {
     "Bio",
"Accuracy",
"Root Mean Squared Error",
"Mean Cross-Entropy",
        "ROC area",
"ROC area up to 50 negative examples",
"Sensitivity",
        "Specificity",
"Negative Predictive Value",
"Positive Predictive Value",
"Precision",
        "Recall",
"F1 score",
"Precision/Recall Break Even Point",
        "Mean Average Precision",
"Lift (at threshold)",
"Top 1: is the top ranked case positive",
"Top 10: is there a positive in the top 10 ranked cases",
"Rank of *last* (poorest ranked) positive case",

"How many positives in the top N ranked cases",
"Norm error using L[arg] metric",
"Set threshold",
/*"Total cost using these cost values, plus min-cost results",
"SAR = (wACC*ACC + wROC*ROC + wRMS(1-RMS))/(wACC+wROC+wRMS)",
"CA1/CA2 scores",*/
"Slac Q-score",
"perfcommand"

        //"Recall",
//"F1 score",
//"Precision/Recall Break Even Point",
        //"Mean Average Precision",
    };

    private static final Class[] INPUT_CLASSES = { ExampleSet.class };
    private static final Class[] OUTPUT_CLASSES = { PerformanceVector.class };
    //private PerformanceVector results = new PerformanceVector();
    private boolean bio;
```

```
    public IOObject[] apply() throws OperatorException {


    PerformanceVector viktor = new PerformanceVector();
    PerformanceVector tempViktor = new PerformanceVector();

    String options="";

     try{
ExampleSet testSet = (ExampleSet)getInput(ExampleSet.class);

bio = getParameterAsBoolean("Bio");

// Benutze ExampleFormatter von Yale den auch ExampleSetWriter von Yale benutzt
String format;
if(bio)
format = "$i $l $p$n";
else
format = "$l $p$n";


ExampleFormatter formatter;
try {
    formatter = ExampleFormatter.compile(format, testSet);
} catch (Throwable e) {
    throw new UserError(this, 901, format, e.getMessage());
}
try {
    PrintWriter out = new PrintWriter(new FileWriter("perf.input"));
    ExampleReader reader = testSet.getExampleReader();
    String tmp;
    StringTokenizer st;
    String line;

    while (reader.hasNext()) {
tmp = formatter.format(reader.next());

// BlockID von double in int wandeln sonst krachts
if(bio) {

st=new StringTokenizer(tmp, " ");
double d = new Double(st.nextToken()).doubleValue();
int dummy = (int) d;
line=dummy+" "+st.nextToken()+" "+st.nextToken();

out.print(line);
}
else
out.print(tmp);
    }
    out.close();
} catch (IOException e) {
    throw new UserError(this, e, 301, new Object[] {"perf.input",e.getMessage()});
}


// wr.close();
//KDDCriterion temp=null;

for (int i=0; i<CRITERIA_NAMES.length; i++) {
options="";
if(i<19){
if (getParameterAsBoolean(CRITERIA_NAMES[i])) {
//if(i==0) options="-BIO";
if(i==1) options="-ACC";
if(i==2) options="-RMS";
if(i==3) options="-CXE";
if(i==4) options="-ROC";
if(i==5) options="-R50";
if(i==6) options="-SEN";
if(i==7) options="-SPC";
if(i==8) options="-NPV";
if(i==9) options="-PPV";
if(i==10) options="-PRE";
if(i==11) options="-REC";
if(i==12) options="-PRF";
if(i==13) options="-PRB";
if(i==14) options="-APR";
if(i==15) options="-LFT";
if(i==16) options="-TOP1";
if(i==17) options="-TOP10";
if(i==18) options="-RKL";
if(getParameter(CRITERIA_NAMES[21])!=null) options+=" -t "+getParameter(CRITERIA_NAMES[21]);
//(if(i==19) options="-
//if(i==17) options+=" -NTOP ";
// Reziproke Werte noch beruecksichtigen
KDDCriterion k;
if(i==3){
```

```
k = new KDDCriterion(CRITERIA_NAMES[i],Double.NaN,true);

}
else{
k = new KDDCriterion(CRITERIA_NAMES[i],Double.NaN,false);

}
k.setOptions(options);
viktor.addCriterion(k);

}
}
else if(i!=23){//alles ausser perfcommand wird auch Kriterium
if(getParameter(CRITERIA_NAMES[i])!=null){
if(i==19) options="-NTOP "+getParameter(CRITERIA_NAMES[i]);
if(i==20) options="-NRM "+getParameter(CRITERIA_NAMES[i]);
if(i==22) options="-SLQ "+getParameter(CRITERIA_NAMES[i]);
if(getParameter(CRITERIA_NAMES[21])!=null) options+=" -t "+getParameter(CRITERIA_NAMES[21]);

if(i!=21){
KDDCriterion k = new KDDCriterion(CRITERIA_NAMES[i],Double.NaN,false);

k.setOptions(options);
viktor.addCriterion(k);

}
}
}

}


tempViktor=perf(viktor);
/


}catch (Exception e){System.out.println("Error:"+e.toString()+"\n"+e.getMessage());}

     // String parameter = getParameterAsString("parameters");
    return new IOObject[] {tempViktor};
    }

    private PerformanceVector perf (PerformanceVector dirk){//String name, String option) {


     PerformanceVector result=new PerformanceVector();
try {
for(int i=0; i<dirk.size(); i++){

if(bio&&(i==0))
i++;
KDDCriterion crit=(KDDCriterion)(dirk.getCriterion(i));


Runtime run=Runtime.getRuntime();
int index=getParameterAsInt("perfcommand");

String  command = commands[getParameterAsInt("perfcommand")];
String console="";
console+=command;
if(bio) console+=" "+crit.getOptions()+" -blocks -file perf.input";
else console+=" "+crit.getOptions()+" -file perf.input";


Process p=run.exec(console);
InputStream is=p.getInputStream();
FileWriter wr =new FileWriter("perf.output");

int byt=is.read();
while(byt!=-1){
wr.write(byt);
byt=is.read();
}
wr.close();
is.close();
FileReader fileReader=new FileReader("perf.output");
LineNumberReader lineReader =new LineNumberReader(fileReader);
String line =lineReader.readLine();
if(line!=null){

StringTokenizer st=new StringTokenizer(line," ");
st.nextToken();
try{
double tempDouble=new Double(st.nextToken()).doubleValue();
//if(crit.getName().equals("Mean Cross-Entropy")) tempDouble=-tempDouble;

if(crit.getName().equals("Mean Cross-Entropy")) crit = new KDDCriterion(crit.getName(),tempDouble,true);
else crit = new KDDCriterion(crit.getName(),tempDouble,false);
```

```
result.addCriterion(crit);


}catch(NumberFormatException e){
System.out.println("No double value:  "+e.toString());
}
}
lineReader.close();
fileReader.close();
}
} catch (Exception e) {System.out.println("Error: "+e.toString()+"\n"+e.getMessage());}

return result;

    }



    public Class[] getInputClasses() {
      return INPUT_CLASSES;
    }

    public Class[] getOutputClasses() {
return OUTPUT_CLASSES;
    }

    public List getParameterTypes() {
List types = super.getParameterTypes();

types.add(new ParameterTypeBoolean("Bio", "Bei Biodaten wird zusaetzlich die ID benutzt", false));
types.add(new ParameterTypeBoolean("Accuracy", "physik", false));
types.add(new ParameterTypeBoolean("Root Mean Squared Error", "bio", false));
types.add(new ParameterTypeBoolean("Mean Cross-Entropy", "physik", false));
types.add(new ParameterTypeBoolean("ROC area", "physik", false));
types.add(new ParameterTypeBoolean("ROC area up to 50 negative examples", "physik.", false));
types.add(new ParameterTypeBoolean("Sensitivity", "physik.", false));
types.add(new ParameterTypeBoolean("Specificity", "physik.", false));
types.add(new ParameterTypeBoolean("Negative Predictive Value", "physik.", false));
types.add(new ParameterTypeBoolean("Positive Predictive Value", "bio.", false));
types.add(new ParameterTypeBoolean("Precision", "bio.", false));
types.add(new ParameterTypeBoolean("Recall", "bio.", false));
types.add(new ParameterTypeBoolean("F1 score", "bio.", false));
types.add(new ParameterTypeBoolean("Precision/Recall Break Even Point", "physik.", false));
types.add(new ParameterTypeBoolean("Mean Average Precision", "bio", false));
types.add(new ParameterTypeBoolean("Lift (at threshold)", "physik.", false));
types.add(new ParameterTypeBoolean("Top 1: is the top ranked case positive", "bio", false));
types.add(new ParameterTypeBoolean("Top 10: is there a positive in the top 10 ranked cases", "physik.", false));
types.add(new ParameterTypeBoolean("Rank of *last* (poorest ranked) positive case", "bio", false));


types.add(new ParameterTypeInt("How many positives in the top N ranked cases","-NTOP [N]",1,1000,false));
types.add(new ParameterTypeDouble("Norm error using L[arg] metric","-NRM [arg]",0.0,1.0,false));
types.add(new ParameterTypeDouble("Set threshold","-t [arg]",0.0,1.0,false));
types.add(new ParameterTypeDouble("Slac Q-score", "physik",0.0,1.0,false));
types.add(new ParameterTypeCategory("perfcommand","Path to the perf executable",commands,2));

//types.add(new ParameterTypeString("parameter", "parameters for perf.", false));
return types;
    }
}
```

Literaturverzeichnis

[1] Kdd cup 2003, 2003. http://www.cs.cornell.edu/projects/kddcup/index.html.

[2] A.Ben-Hur, D.Horn, H.T.Siegelmann, and V.Vapnik. Support vector clustering. *Journal of Maschine Learning Research*, 2:125–137, 2001.

[3] Albrecht Achilles. *SQL: standardisierte Datenbanksprache vom PC bis zum Mainframe; von dBASE IV bis zu DB2 und SQL/DS.* Oldenbourg Wissenschaftsverlag GmbH, München, 2000. 7. Auflage.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large data bases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*, pages 478–499, Santiago, Chile, sep 1994.

[5] Tej Anand and Ronald J. Brachman. The process of knowledge discovery in databases: A human-centered approach. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, AAAI Press Series in Computer Science. A Bradford Book, The MIT Press, Cambridge Massachusetts, London England, 1996.

[6] E. Andrassyova and J. Paralic. Knowledge discovery in databases: A comparison of different views. http://neuron-ai.tuke.sk/ andrassy/publikacie/varazdin/andrassy.doc.

[7] Leo Breiman. Bagging predictors. Technical Report 421, Department of Statistics, Univerity of California, Berkley, California 94720, September 1994.

[8] Leo Breiman. Heuristics of insatibility in model selection. Technical report, Statistics Department, University of California at Berkeley, 1994.

[9] Leo Breiman and Spector P. Submodel selection and evaluation in regression - the x-random case. *International Review of Statisctics*, 3:291 –319, 1992.

[10] Peter Brockhausen and Katharina Morik. Direct access of an ILP algorithm to a database management system. In Johannes Fürnkranz and Bernhard Pfaringer, editors, *Data Mining with Inductive Logic Programming (ILP for KDD)*, MLnet Sponsored Familiarization Workshop, pages 95–110, Bari, Italy, jul 1996.

[11] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[12] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. http://www.cs.cornell.edu/ caruana/caruana.icml04.crc.ps.

[13] Pete Chapman, Julian Clinton, Thomas Khabaza, Thomas Reinartz, and Rüdiger Wirth. The crisp–dm process model. Technical report, The CRIP–DM Consortium NCR Systems Engineering Copenhagen, Daimler-Chrysler AG, Integral Solutions Ltd., and OHRA Verzekeringen en Bank Groep B.V, March 1999. This Project (24959) is partially funded by the European Commission under the ESPRIT Program.

[14] J. Cheng, C. Hatzis, H. Hayashi, M.-A. Krogel, S. Morishita, D. Page, and J. Sese. Kdd cup 2001 report. *SIGKDD Explorations Volume3 Issue2*, pages 47–64, 2002. http://www.acm.org/sigkdd/explorations/issue3-2.htm.

[15] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[16] Edgar F Codd. Extending the database relational model to capture more meaning. *ACM Transactiions on Database Systems*, 4(4):397–434, dec 1979.

[17] Edgar F Codd. *The relational model for database management version 2.* Addison-Wesley, Reading, Mass., 1990.

[18] Luc De Raedt and Stephen Muggleton. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[19] D.Fasulo. An analysis of recent work on clustering algorithms. Technical report, University of Washington, Department of Computer Science and Engineering, 1999.

[20] D.Gibson, J.Kleinberg, and P.Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Proceedings of the 24th VLDB Conference, New York, USA*, 1998.

[21] J. Diez, J.J. del Coz, O.Luaces, and A.Bahamonde. A clustering algorithm to find groups with homogeneneous preferences.

[22] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *Interantional Journal of Pattern Recognition and Artificial Intelligence*, 4(7):705 – 719, 1993.

[23] B. Efron and R.J. Tibshirani. *An Introduction to the Boostrap*. Chapman & Hall, 1993.

[24] Inger et al. Kdd-cup 2000: Question 1 winner's report. *SIGKDD Explorations*, 2000.

[25] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[26] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–36. AAAI Press/The MIT Press, Menlo Park, California, 1996.

[27] Usama Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press Series in Computer Science. A Bradford Book, The MIT Press, Cambridge Massachusetts, London England, 1996.

[28] Doug Fisher. Iterative optimization and simplification of hierarchical clusterings. *Artificial Intelligence Research*, 4:147–179, apr 1996.

[29] Yoav Freund. Boosting a weak learning algorithm by majority. *Inforamtion and Computation*, 121(2):256 – 285, 1995.

[30] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 –139, August 1997.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. Technical report, Departement of Statistics, Stanford University, Stanford, California 94305, July, 23 1998.

[32] Norbert Fuhr. Information retrieval. http://www.is.informatik.uni-duisburg.de/teaching/lectures/db-ws02/folien/irfk12.pdf.

[33] Joao Gama and Pavel Brazdil. Constructive induction on continuous spaces. In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer, 1998.

[34] Friedrich Gebhardt. Choosing among competing generalizations. *Knowledge Acquisition*, pages 361–380, 1991.

[35] G. Goerz, C.-R. Rollinger, and J. Schneeberger (Hrsg.). *Handbuch der kuenstlichen Intelligenz*. Oldenburg Verlag, 2000.

[36] Bart Goethals. *Survey on Frequent Pattern Mining*. Department of Computer Science, University of Helsinki, 2003.

[37] Günther Görz, editor. *Einführung in die künstliche Intelligenz*. Addison-Wesley, 2 edition, 1995.

[38] Han, Pei, and Yin. Minig frequent patterns without candidate generation. *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–8, 1999.

[39] David J. Hand. Statistics and data mining: Intersecting disciplines. *SIGKDD Explorations*, 1(1):16–17, 1999.

[40] D. R. Howe. *Data Analysis for Data Base*. Edword Arnold, 1983.

[41] Michael R.A Huth and Mark D. Ryan. *Logic in Computer Science, Modelling and reasoning about systems*. Cambridge University Press, 2000. Reprint 2001.

[42] Anne H. Milley Jim Georges. Kdd'99 competition: Knowledge discovery contest. Technical report, SAS Institute Inc., Cary, NC 27513-2414, USA, 1999.

[43] Thorsten Joachims. Optimizing search engines using clockthrough data. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[44] Miachael Kearns and Leslie G. Valinat. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 1(41):67 – 95, Januar 1994.

[45] Miachael Kearns and Leslie G. Valinat. Learning boolean formulae or finite automata is as hard as factoring. Technical report tr-14-88, Havard University Aiken Computation Laboratory, August 1988.

[46] Willi Kloesgen. Explora: A multipattern and multistrategy discovery assistant, 1996.

[47] Willi Kloesgen. *Change analysis.* 2000.

[48] R. Kohavi, C.E.Brodley, B.Frasca, L.Mason, and Z.Zheng. Kdd-cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations Volume2 Issue2*, pages 86–98, 2000. http://www.acm.org/sigkdd/explorations/issue2-2.htm.

[49] Guido Lindner. Anwendung des Lernverfahrens RDT auf eine relationale Datenbank. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, Lehrstuhl VIII, aug 1994. Eine Zusammenfassung ist auch als Forschungsbericht Nr. 12 des Lehrstuhls Informatik VIII, Universität Dortmund, erschienen.

[50] H. Liu and H. Motoda. *Feature Extraction, Construction, and Selection: A Data Mining Perspective.* Kluwer, 1998.

[51] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–290, November 1997.

[52] Thomas Mitchell. *Machine Learning.* McGraw Hill, 1997.

[53] Katharina Morik and Martin Scholz. The MiningMart Approach. In *Workshop Management des Wandels der 32. GI Jahrestagung*, 2002.

[54] Dirk Münstermann. Wissensentdeckungen in datenbanken mit dynamischer anpassung des hypothesentests. Diplomarbeit, Fachbereich Informatik, Universität Dortmund, 2002.

[55] MySQL AB. *MySQL Reference Manual for version 4.0.11 gamma (2003)*, 2003.

[56] J. Shawe-Taylor N. Cristianini. *An introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.

[57] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler, Band 3.* Vieweg, 1997.

[58] B. Pfahringer. First winner report. *SIGKDD Explorations Volume1 Issue2*, pages 65–66, 2000. http://www.acm.org/sigkdd/explorations/issue1-2.htm.

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

[60] R.Agrawal, J.Gehrke, D.Gunopulus, and P.Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD'98, Seatle, USA*, 1998.

[61] P.Hart R.Duda. *Pattern Classification And Scene Analysis.* A Wiley-Interscience Publication, USA, 1973.

[62] Robert Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197 – 227, 1990.

[63] Robert Schapire. The boosting approach to machine learning - an overview. *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[64] Robert Schapire, Yoav Freund, Peter BArlett, and Wee Sung Lee. Boosting the margin: A new explanation for for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651 – 1686, October 1998.

[65] Qingzhao Tan, Xiaoyong Chai, Wilfred Ng, and Dik-Lun Lee. Applying co-training to clickthrough data for search engine adaption.

[66] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[67] Gottfried Vossen. *Datenbank-Theorie.* Thomson Publishing, Bonn, 1995. 1. Auflage.

[68] Hubert Wagner. Logische systeme der informatik. Skriptum, Fachbereich Informatik, Universitäet Dortmund, 2000.

[69] Ian H. Witten and Eibe Frank. *Data Mining.* Hanser, 2001.

[70] Stefan Wrobel. An algorithm for multi–relational discovery of subgroups. In J. Komorowski and Jan Zytkow, editors, *Principles of Data Minig and Knowledge Discovery: First European Symposium (PKDD 97)*, pages 78–87, Berlin, New York, 1997. Springer.

[71] A. Yeh, L. Hirschman, A. Morgan, and M. Craven. Reports from kdd-2002. *SIGKDD Explorations Volume4 Issue2*, pages 87–109, 2003. http://www.acm.org/sigkdd/explorations/issue4-2.htm.