Diplomarbeit

Universität Rostock

Fachbereich Informatik

Ein Meta-Lerner zur automatischen Auswahl von Data-Mining-Verfahren

eingereicht von Cornelia Laudien
geboren 13.04.1977 in Rostock
Betreuer: Adelinde Uhrmacher, Andreas Heuer, Meike Klettke
Abgabetermin: 6. November 2001

Zusammenfassung

Aufgrund der zunehmenden Vielfalt von Data-Mining-Verfahren steigt das Interesse an Systemen, die Hilfestellung bei der Auswahl eines adäquaten Verfahrens zur Datenanalyse geben. Ziel dieser Diplomarbeit ist es, ein tieferes Verständnis für die verschiedenen Methoden des Data Mining zu entwickeln und ein Konzept zu entwerfen, das zur Automatisierung des Auswahlprozesses beiträgt. Es wird ein Meta-Lerner allgemein konzipiert, der Beziehungen zwischen Problemstellungen und der Performanz einzelner Algorithmen herstellt. Dieser wird dann für die Auswahl von Klassifikationsverfahren konkretisiert. Im Gegensatz zu anderen Meta-Lernern wird hier ein neuerer Ansatz zur Problembeschreibung auf der Basis von Entscheidungsbäumen aufgegriffen. Der Meta-Lerner ist als fallbasiertes System implementiert, welches zur Gewichtung der Attribute ein informationstheoretisches Maß nutzt. Experimente demonstrieren, dass der gewählte Ansatz erfolgversprechend ist.

Abstract

Because of the increasing variety of data mining methods there is a rising interest in systems assisting in the selection of an adequate method for data analysis. The goal of this master's thesis is to support a deeper understanding of the different data mining methods and to develop a concept contributing to the automation of the selection process. A general architecture relating tasks and algorithm performance has been developed. A meta learner for classification has been implemented as an instance of this architecture. Contrary to other meta learners a novel way of describing tasks on the basis of decision trees has been adopted. The meta learner realizes a case based reasoning system that uses a information theoretic measure to weight attributes. Experiments demonstrate that the chosen strategy is very promising.

CR-Klassifikation

H.2.8 Data Mining
I.2.6 Learning

Key Words

Knowledge Discovery, Data analysis, Data Mining, Machine Learning, Model Selection, Meta-Learning, Case Based Reasoning, Classification, Metadata

Inhaltsverzeichnis

1 Einleitung und Motivation

In den Datenbanken der heutigen Unternehmen werden immer mehr Daten angehäuft. Damit aus diesen Daten Informationen gewonnen werden können, sind Systeme notwendig, die eine Analyse dieser Daten ermöglichen. Von diesen Systemen wird erwartet, dass sie neue Beziehungen zwischen Variablen und verborgene Datenmuster erkennen können. *Knowledge Discovery in Databases* (KDD) ist der nicht-triviale Prozess der Identifikation gültiger, neuer, potentiell nützlicher und verständlicher Muster in Datenbanken, [FPSSU96]. Der KDD-Prozess, das Entdecken von Wissen, gliedert sich in folgende Phasen:

1. Verstehen des Problembereiches und Bestimmen der Ziele des KDD-Prozesses aus Sicht des Kunden
2. Bestimmen der notwendigen Ressourcen und Festlegen des Datensatzes, der analysiert werden soll
3. Datenbereinigung und -vorverarbeitung, z.B. Finden einer Strategie zur Handhabung fehlender Werte
4. Datenreduktion und -projektion, z.B. Anwenden von Transformationsmethoden, um eine beständige Darstellung der Daten zu gewinnen
5. Zuordnen der Ziele des KDD-Prozesses (siehe 1.) zu einer bestimmten Data-Mining-Aufgabe, z.B. Klassifikation
6. Auswählen eines geeigneten Data-Mining-Algorithmus' entsprechend der Aufgabe und Festlegen seiner Parameter
7. Ausführen des ausgewählten Algorithmus'
8. Bewertung der gefundenen Muster und gegebenenfalls Wiederholen vorheriger Schritte
9. Nutzen des gefundenen Wissens

Im Gegensatz zu Projekten, wie CITRUS (siehe [WSG+97]), die versuchen, dem Nutzer in allen Phasen des KDD-Prozesses zu assistieren, liegt der Schwerpunkt dieser Diplomarbeit in einem Teilschritt des Prozesses, nämlich in der Unterstützung des Nutzers bei der Auswahl des Data-Mining-Algorithmus'.
Im nächsten Kapitel werden einige Algorithmen vorgestellt und den einzelnen Data-Mining-Aufgaben zugeordnet. Die Vielzahl der dargestellten Algorithmen soll zeigen, dass die Auswahl einer geeigneten Data-Mining-Methode auch für Experten auf dem Gebiet des Maschinellen Lernens eine schwierige Aufgabe ist. Bisher wurde dieses Problem durch Experimentieren oder durch Konsultation eines Experten gelöst. Die erste Variante ist sehr zeitaufwendig und unzuverlässig und die zweite ist teuer und von den Präferenzen des Experten abhängig. Ziel dieser Arbeit ist es, ein tieferes Verständnis für die verschiedenen Methoden des Data Mining zu entwickeln und ein Konzept zu entwerfen, das zur Automatisierung des Auswahlprozesses beiträgt.

Im dritten Kapitel wird auf die Idee des Meta-Lernens eingegangen und wie sie zur automatischen Methodenauswahl eingesetzt werden kann. In diesem Kapitel finden sich viele der im Rahmen des METAL-Projektes [Met01] veröffentlichten Ergebnisse. Ziel des Projektes ist die Entwicklung eines prototypischen Assistenzsystems, das den Nutzer bei der Verfahrensauswahl und Methodenkombination speziell für die Klassifikation unterstützen soll. Die in diesem Kapitel vorgestellten Konzepte geben die Impulse für den in Kapitel 4 konzipierten Meta-Lerner. Dieser Meta-Lerner wird es Nutzern, die keine Experten auf dem Gebiet des Maschinellen-Lernens sind, ermöglichen, einen geeigneten Data-Mining-Algorithmus auszuwählen und seine Parameter festzulegen.
Eine prototypische Implementierung des Meta-Lerners speziell für die Empfehlung von Klassifikationsverfahren ist Bestandteil dieser Arbeit und wird im Kapitel 5 beschrieben. Die Darstellung der zur Evaluierung des entstandenen Prototypen durchgeführten Experimente sind ebenfalls Bestandteil dieses Kapitels. Die Arbeit wird durch eine Zusammenfassung der Ergebnisse abgeschlossen.

2 Gebräuchliche Verfahren des Data Mining

Data Mining ermöglicht die Entdeckung interessanter Zusammenhänge in sehr großen Datenbeständen. Fayyad bezeichnet in [FPSSU96] *Data Mining* als einen Schritt im KDD-Prozess bestehend aus bestimmten Data-Mining-Algorithmen, die unter akzeptierbaren rechnerischen Einschränkungen der Effizienz, eine besondere Aufzählung von Mustern über Daten erzeugt.

Die meisten Data-Mining-Methoden basieren auf Konzepten des Maschinellen Lernens, Mustererkennung und Statistik. Diese Methoden können den Nutzer bei der Generierung von Hypothesen unterstützen. Des weiteren sei darauf hingewiesen, dass nicht nur die Konzepte für sich genommen Data Mining ausmachen, sondern auch die Tatsache, dass die damit verbundenen Verfahren auch effizient und skalierbar auf großen Datenmengen anwendbar sind. Diese Problematik wird ausführlich in [Rei99] diskutiert.

Die dabei in der Praxis verfolgten Hauptziele sind die Vorhersage/Prognose und die Beschreibung. Die Vorhersage nutzt bekannte in der Datenbank enthaltene Attribute, um unbekannte oder zukünftige Werte von anderen Attributen vorherzusagen. Die Beschreibung hingegen versucht, vom Menschen interpretierbare Muster, die in den Daten enthalten sind, zu erfassen.
In [HS91] wird Data Mining als eine besondere Art des Maschinellen Lernens bezeichnet, wobei die Umgebung, in der gelernt wird, durch eine Datenbank repräsentiert wird. Durch das Anwenden von Data-Mining-Verfahren werden auf der einen Seite Datenbank Management Systeme (DBMS) um induktive Fähigkeiten erweitert. Auf der anderen Seite müssen die Verfahren des Maschinellen Lernens derart erweitert werden, dass sie mit den in Datenbanken möglichen unvollständigen, redundanten und verrauschten Daten umgehen können.
Bensusan stellt in [Ben99] Induktion als Hauptantrieb der Wissensakquisition heraus. Dabei versteht er Induktion als Produktion allgemeiner Schlussfolgerungen bzw. Hypothesen aus bestimmten Daten oder Beobachtungen. Beim induktiven Lernen werden zwei Lerntechniken unterschieden:

- Zum einen gibt es das überwachte Lernen, wobei ein Lehrer Klassen definiert und dem System Trainingsbeispiele für jede Klasse vorgibt. Ein Beispiel stellt dabei eine Menge von Attributausprägungen zusammen mit der zugeordneten Klasse dar. Das System hat nun die Aufgabe, Gemeinsamkeiten innerhalb der Beispiele einer Klasse und Unterschiede zwischen den einzelnen Klassen zu finden. Aufgrund von gefundenen Gemeinsamkeiten und Unterschieden kann eine Klassenbeschreibung erstellt werden, die genutzt wird, um neue Datensätze richtig einzuordnen bzw. Attributwerte vorherzusagen. Durch Anwenden der Methoden des überwachten Lernens kann also eines der oben erwähnten Hauptziele, die Vorhersage, erreicht werden.
- Zum anderen gibt es das unüberwachte Lernen. Hierbei muss das System selbst aufgrund von Zusammenhängen oder Auffälligkeiten Klassen in den

Daten entdecken. Das heißt, Methoden des unüberwachten Lernens liefern eine Beschreibung der Daten. Das zweite oben erwähnte Hauptziel ist damit erreicht.

Allgemein kann man sagen, etwas lernt, wenn es sein Verhalten so ändert, dass es in Zukunft eine bessere Leistung aufweist. In [WF01] geht man noch einen Schritt weiter und spricht von vier grundsätzlich verschiedenen Arten des Lernens im Data-Mining-Bereich:

- dem klassifizierenden Lernen (Klassifikation),
- dem assoziierenden Lernen (Assoziation),
- dem Clustering und
- der numerischen Vorhersage (Regression)

Die numerische Vorhersage umfasst jedoch einen Spezialfall der Klassifikation, so dass sich Klassifikation, Assoziation und Clustering als die drei wesentlichen Verfahrensfamilien des Data Mining herausstellen. Die Methoden des klassifizierenden Lernens und der numerischen Vorhersage werden dem überwachten Lernen zugeordnet, während das assoziierende Lernen und das Clustering zum unüberwachten Lernen gehören.

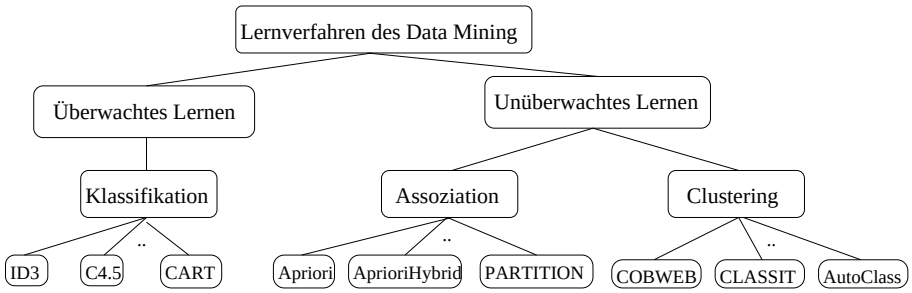


Abbildung 1: Gebräuchliche Lernverfahren des Data Mining

Zusammenfassend lässt sich feststellen, dass die gebräuchlichen Data-Mining-Verfahren Lernverfahren sind, die überwacht oder unüberwacht ablaufen können, so dass sie entweder Vorhersagemodelle oder Beschreibungen finden. Im folgenden werden die Begriffe Data-Mining-Verfahren und Lernverfahren synonym verwandt.
Weiterhin können Data-Mining-Verfahren danach unterschieden werden, auf welche Art sie ein Vorhersagemodell bzw. eine Beschreibung erstellen. Dies kann

sowohl inkrementell als auch oneshot-artig sein, darauf soll an dieser Stelle jedoch nicht näher eingegangen werden. Abbildung 1 stellt die gefundenen Zusammenhänge noch einmal dar. Die konkreten Algorithmen, wie z.B. ID3 oder Apriori, werden zusammen mit ihrer Verfahrensfamilie in den folgenden Unterabschnitten erläutert.

2.1 Klassifikation

Klassifikation beinhaltet das Lernen einer Konzeptbeschreibung, die es ermöglicht, Datensätze zu vordefinierten Klassen zuzuordnen. Die Wissensrepräsentation kann dabei in Form von Regeln oder eines Entscheidungsbaumes erfolgen. Da dem Lernverfahren vorgegebene Klassifikationen für alle Trainingsdatensätze zur Verfügung gestellt werden, ist es ein Verfahren des überwachten Lernens. Der Erfolg des klassifizierenden Lernens kann beurteilt werden, indem man die gelernte Zuordnungsfunktion für eine unabhängige Testdatenmenge ausprobiert, wobei die richtigen Klassen im Voraus bekannt sind, aber dem System nicht zur Verfügung gestellt werden. Auf die Problematik der Auswahl von Trainings- und Testdatensätzen wird in einem späterem Kapitel noch genauer eingegangen.

ID3: Am bedeutendsten von allen Klassifikationsalgorithmen ist wohl ID3 (Induction of Decision Trees), der in den ’80er Jahren von Quinlan entwickelt wurde. Viele der heute eingesetzten Systeme nutzen Variationen dieser Methode. [Her97] bezeichnet ID3 als “Klassiker” der Lernsysteme. Anhand einer Menge von Attributen, deren unterschiedliche Ausprägungen als Unterscheidungsmerkmale dienen, wird ein *Entscheidungsbaum* aufgebaut. Im allgemeinen verfolgen Entscheidungsbäume einen “Teile und Herrsche”-Ansatz. Man sucht auf jeder Ebene ein Attribut für die Zerlegung und verarbeitet rekursiv die aus der Zerlegung resultierenden Teilprobleme. Jeder interne Knoten des Baumes stellt dabei einen Test auf die Werte eines Attributes dar. Die Verzweigungen auf Nachfolgerknoten entsprechen den verschiedenen Attributwerten. Die Blattknoten spezifizieren die Klassen, die gelernt werden sollen. ID3 führt den Lernprozess als eine Top-Down-Suche durch, d.h. der Entscheidungsbaum wird schrittweise spezialisiert, indem er neue Verzweigungen erhält. Deshalb wird ID3 auch als TDIDT-Lernsystem bezeichnet, wobei TDIDT für “Top-Down Induction of Decision Trees” steht. Der Baum wird wie folgt aufgebaut:

1. Ein Attribut wird als Wurzelelement ausgesucht. Für jeden Wert, den dieses Attribut annehmen kann, wird ein Zweig angelegt.

2. Die Trainingsdaten werden nun anhand dieses Baumes klassifiziert. Wenn alle Beispiele eines Blattes zur gleichen Klasse gehören, wird das Blatt mit dieser Klasse gekennzeichnet. Wenn alle Blätter mit einer Klasse gekennzeichnet sind, bricht der Algorithmus ab.

3. Andernfalls wird der Knoten mit einem Attribut gekennzeichnet, das auf dem bisherigen Pfad zur Wurzel noch nicht erscheint und es wird wieder ein Zweig für jeden möglichen Attributwert angelegt. Zurück zu 2.

Damit jedoch der effizienteste Entscheidungsbaum gefunden wird, ist die richtige Auswahl der Attribute entscheidend. Mit dem oben beschriebenen Algorithmus findet man einen Baum, der die Trainingsmenge korrekt klassifiziert, aber es ist nicht unbedingt der einfachste Baum. Deshalb wendet ID3 eine Heuristik an, die dasjenige Attribut auswählt, welches den Informationsgewinn maximiert, siehe dazu [Her97].

C4.5: Diverse Verbesserungen von ID3 führten zu einem einflussreichen und weit verbreiteten System, dem C4.5. Hierbei wird aus einem generierten *Entscheidungsbaum* eine *Regelmenge* erzeugt und zusätzlich eine *Postpruning-Methode* angewandt. Postpruning ist eine Technik, die das Überanpassen des Entscheidungsbaumes bzw. der Klassifikationsregeln an die Trainingsmenge verhindert. Eine Überanpassung führt gerade bei verrauschten Daten oder bei einer zu kleinen Anzahl von Trainingsbeispielen zu Problemen. Die abgeleitete Regelmenge hat die Form: IF <Bedingungen> THEN <Hypothese>, wobei die Bedingungen Attribut-Wert-Paare darstellen. Eine Hypothese ist im Fall des C4.5 Algorithmus' genau ein Attribut-Wert-Paar. Jede Klassifikationsregel ist durch ein Genauigkeitsmaß gekennzeichnet, welches angibt, wieviele Instanzen einer Klasse durch diese Regel abdeckt werden. Dieses Maß drückt die Qualität einer Regel aus.
Die Vorgehensweise des C4.5 kann grob in vier Schritten beschrieben werden:

1. Zunächst wird ein Entscheidungsbaum aus den Trainingsbeispielen abgeleitet, wobei Überanpassung zugelassen wird.

2. Dann erfolgt die Überführung des gelernten Baumes in eine äquivalente Regelmenge, wobei eine Regel pro Pfad von der Wurzel zu einem Blatt erzeugt wird.

3. Jetzt werden die Regeln verallgemeinert (Pruning) durch Streichen bestimmter Bedingungen. Wie man die zu entfernenden Bedingungen herausfindet, ist in [Mit97] im Zusammenhang mit Postpruning, genauer gesagt mit Regel-Postpruning, beschrieben.

4. Zuletzt werden die verallgemeinerten Regeln nach ihrer geschätzten Genauigkeit sortiert, damit sie später bei der Klassifikation der Daten in dieser Reihenfolge berücksichtigt werden.

Daraus ergibt sich, dass C4.5 besser mit fehlenden Werten und verrauschten Daten umgehen kann als ID3. Der Nachfolger C5.0 (siehe [C50]) zeichnet sich im Vergleich zu C4.5 dadurch aus, dass er wesentlich schneller die Regelmenge aus dem

Entscheidungsbaum ableiten kann.
Andersartige Weiterentwicklungen des ID3 sollen an dieser Stelle noch kurz genannt werden. Zum Beispiel wurde der GID3 Algorithmus von Fayyad entwickelt (siehe [FPSSU96]). Er ist in der Lage Verzweigungen eines Baumknoten nur für ausgewählte Attributwerte anzulegen. Die restlichen Attributwerte werden durch einen speziellen Zweig zusammengefasst repräsentiert. Ein anderer, der O-BTree Algorithmus, generiert ausschließlich binäre Bäume mit Hilfe des ORT-Masses (orthogonality measure), um den Aufwand der Attributauswahl zu verringern. Dieser Algorithmus wurde von Fayyad in Zusammenarbeit mit Irani entwickelt (siehe [FI92]).

AQ15: Michalski's AQ15 (siehe [MBK01]) lernt eine Menge von Wenn-Dann-Regeln. Allerdings wird die Regelmenge direkt gelernt und nicht wie bei C4.5, der zunächst einen Entscheidungsbaum lernt und diesen dann in eine Regelmenge übersetzt. Außerdem lernt AQ15 Regeln erster Ordnung, d.h. die Regeln können Variablen enthalten und sind dadurch viel ausdrucksstärker. Diese Regeln werden auch als *Horn-Klauseln* erster Ordnung bezeichnet und können als Programme der logischen Programmiersprache Prolog interpretiert werden. Deshalb wird das Lernen dieser Regeln oft als induktive logische Programmierung (ILP) bezeichnet. Ein weiterer Vertreter der induktiven logischen Programmierung ist FOIL (siehe [Her97]). Die Konstruktion von Regeln wird auch als *Abdeckungsalgorithmus* (Covering) bezeichnet. AQ15 verwendet die Top-Down-Suche, um für jede Klasse eine Regel zu erzeugen. Er beginnt also mit der allgemeinsten Regel, die möglichst viele der in der Klasse enthaltenen Instanzen abdeckt und gleichzeitig Instanzen ausschließt, die nicht in der Klasse enthalten sind. Die Regel wird solange mit Klauseln ergänzt, bis sie alle positiven und keine negativen Instanzen abdeckt. Eine genauere Beschreibung des Algorithmus', sowie Informationen zu einer Weiterentwicklung von AQ15, namens CN2, können in [HS91] nachgelesen werden. AQ15 selbst kann nicht mit verrauschten Daten umgehen, erst durch Vor- und Nachbearbeitungsschritte wird das ermöglicht. CN2 überwindet das Problem, indem ähnlich wie in C4.5 Pruning-Techniken integriert wurden.

Naive Bayes: Die Naive Bayes Methode basiert, wie der Name schon sagt, auf *Bayes' Regel* der bedingten Wahrscheinlichkeit. Allerdings geht diese Methode von der Unabhängigkeit der verwendeten Attribute aus (gegeben ihre Zugehörigkeit) und wird deshalb als naiv bezeichnet. Trotz der *Unabhängigkeitsannahme*, die in realen Datenbeständen meistens nicht gegeben ist, arbeitet Naive Bayes laut [HK00b] auch für reale Datenmengen gut. Ein Vorteil ist, dass fehlende Werte kein Problem darstellen, das entsprechende Attribut wird bei der Berechnung einfach weggelassen. Die zur Berechnung herangezogenen Wahrscheinlichkeitsverhältnisse basieren nur auf der Anzahl der tatsächlich vorkommenden Werte und nicht auf der Gesamtanzahl der vorhandenen Datensätze.

CART: Das CART-System (Classification And Regression Trees) von Breiman (siehe [CAR]) umfasst unter anderem ein Verfahren zur Induktion von *Regressionsbäumen*. Regressionsbäume sind Bäume, die zur numerischen Vorhersage verwendet werden. Wie zuvor erwähnt, stellt die numerische Vorhersage einen Spezialfall der Klassifikation dar. Dabei wird nicht ein Konzept für eine vorgegebene Kategorie wie bei den Entscheidungsbäumen von ID3 gelernt, sondern es wird ein Baum erstellt, dessen Blätter einen gemittelten numerischen Wert enthalten.

M5: Der M5-Algorithmus erstellt einen *Modellbaum* (siehe [WF01]). Diese Art der Bäume wird wie die Regressionsbäume des CART-Systems zur numerischen Vorhersage verwendet. Ein Modellbaum ist eine Kombination aus Entscheidungsbaum und linearer Regression, d.h. in jedem Blatt wird ein lineares Regressionsmodell gespeichert, das den Klassenwert von Datensätzen vorhersagt, die das Blatt erreichen. Die aus der Statistik kommende Methode der linearen Regression versucht, die Klasse als lineare Kombination der Attribute mit vorher ermittelten Gewichtungen auszudrücken. Diese Methode kann auch, wenn die Klasse und alle Attribute numerisch sind, als eigenständige Klassifikationsmethode ohne Kombination mit Entscheidungsbäumen eingesetzt werden.

Weitere Klassifikationsmethoden: Es existieren noch weitaus mehr Klassifikationsalgorithmen, z.B. das Lernen mit RBF (Radial Basis Functions) (siehe [Mit97]). Viele Lernalgorithmen entstehen durch Kombination von oben genannten Verfahren, wie z.B. der M5-Algorithmus. Andere werden weiterentwickelt, um genauere Vorhersagen treffen zu können, dabei werden Techniken verwandt, wie:

Bagging ist ein Verfahren bei dem zunächst mehrere Modelle eines Lernverfahrens parallel erzeugt werden, z.B. mehrere Entscheidungsbäume. Dafür wird die Trainingsmenge entsprechend aufgeteilt bzw. sie wird durch Bootstrap Methoden aufbereitet (Bootstrap-Aggregation). Bootstrap verwendet die statistische Prozedur der Stichprobenerfassung mit Ersetzen. Die verschiedenen Ausgaben werden dann zu einem einzigen Modell verschmolzen, dies kann z.B. durch Mehrheitsentscheidung geschehen.

Boosting ist wie Bagging ein Verfahren, bei dem mehrere Modelle erzeugt und kombiniert werden, hier erfolgt die Modellgenerierung jedoch iterativ. Jedes Modell wird durch die Leistung der vorher erzeugten Modelle beeinflusst. Boosting sucht nach Modellen, die sich gegenseitig ergänzen. (C5.0boost, AdaBoost.M1)

Stacking versucht Modelle unterschiedlicher Lernverfahren zu kombinieren. Die Stacking-Methode verfolgt dabei das Konzept eines Metalernsystems, welches das Abstimmungsverfahren von Bagging ersetzt. Es versucht damit die optimale Kombination der Modelle der Basislernsystemen herauszufinden (siehe [FSC]).

Des weiteren schreibt [WF01] von inkrementellen Versionen von Entscheidungsbaum-Induktionsverfahren und Regellernverfahren, die beim Aufbau des Modells immer nur eine Instanz verarbeiten.
Nachdem nun die Vielfalt von Klassifikationsverfahren anhand von einigen ausgewählten Algorithmen vorgestellt wurden, soll es nun um eine weitere Verfahrensfamilie des Data Mining gehen, die Assoziation.

2.2 Assoziation

Das assoziierende Lernen versucht Abhängigkeiten zwischen den Attributen der Datensätze zu erkennen. Das Ergebnis kann entweder in Form von Assoziationsregeln oder Wahrscheinlichkeitsnetzen dargestellt werden. Im Data-Mining-Bereich werden vorwiegend Algorithmen eingesetzt, die Assoziationsregeln generieren. Die Form dieser Regeln unterscheidet sich von Klassifikationsregeln dadurch, dass mehrere Attribute und nicht nur eine einzige Klasse auf der rechten Seite der Regel auftauchen können (siehe [FPSSU96]). Assoziationsregeln werden häufig in Situationen eingesetzt, in denen die Attribute unär sind, also vorhanden oder nicht vorhanden sind. Vorhandene Attribute bzw. in der Regel auftauchende Attribut-Wert-Paare werden in der Literatur als Item (Artikel) bezeichnet, denn Assoziationsregeln werden vorwiegend für Warenkorbanalysen eingesetzt und die Attribute stellen dann die eingekauften Artikel dar. Zusammen auftretende Items werden Transaktionen genannt.
Die Relevanz von Assoziationsregeln wird durch zwei Kennzahlen beschrieben:

- Der Support drückt aus, wieviele Datensätze diese Regel unterstützen, gemessen an der Gesamtanzahl von Datensätzen.

- Die Konfidenz beschreibt den Anteil derjenigen Transaktionen, die beide Seiten der Regel wahr machen, gemessen an denen, die insgesamt die linke Seite der Implikation erfüllen.

Die Erzeugung von Assoziationsregeln gestaltet sich schwieriger als bei Klassifikationsregeln, da weder die linke noch die rechte Seite vorher bekannt ist; das assoziierende Lernen ist ein unüberwachtes Lernen. Um derartige Regeln erzeugen zu können, durchlaufen alle Algorithmen zwei Etappen:

1. Als erstes werden die Transaktionen nach häufig auftretenden Itemmengen durchsucht, die auch als Kandidaten bezeichnet werden. Die Menge aller Kandidaten wird weiter unten mit L bezeichnet.

2. Im zweiten Schritt werden dann die Assoziationsregeln für diese Kandidaten erzeugt.

Die Gesamtanzahl der zu erzeugenden Regeln wird dadurch eingeschränkt, indem die Regeln einem vorgegebenem Minimum-Support *minsup* und einer Minimum-Konfidenz *minconf* genügen müssen. Es wurden viele Algorithmen zum Bestimmen der häufigen Itemmengen entwickelt, da dieser erste Schritt der aufwendigere von beiden ist.

Apriori: Der Apriori Algorithmus ist einer der in der Praxis am häufigsten eingesetzten Algorithmen zum Finden von häufigen Itemmengen. Seine Vorgehensweise kann wie folgt beschrieben werden:
Apriori kombiniert im Grunde die Breitensuche mit dem Zählen des Auftretens von Kandidaten ([HGN00]). Beim ersten Durchlauf der Transaktionen werden zunächst nur die Vorkommen der einzelnen Items gezählt, um die Menge der häufigsten Items entsprechend *minsup* zu bestimmen. Diese Menge wird mit L_1 bezeichnet und besteht zunächst aus einelementigen Kandidatenmengen. Ein späterer Durchlauf n besteht dann aus zwei Phasen. Als erstes werden die $L_n - 1$ Mengen, die im Durchlauf n-1 bestimmt wurden, genutzt, um mit Hilfe der Apriori-Gen Funktion (siehe [FPSSU96]), die neuen Kandidaten von n zu generieren. Danach wird die Datenbank erneut durchlaufen, um den Support dieser Kandidaten zu ermitteln. Wenn keine Kandidaten, die *minsup* genügen, gefunden werden, kann zum zweiten Punkt der Regelgenerierung übergegangen werden.

Erzeugen der Regeln: Wenn alle Supportwerte der Kandidaten bestimmt wurden, können mögliche Regeln generiert und ihre Konfidenz ermittelt werden. Dazu wird jeder Kandidat einzeln betrachtet. Jede seiner Teilmengen wird jeweils als linke Seite einer Regel festgelegt und die übrigen Items bilden die rechte Seite der Regel. Da der Kandidat als häufig vorkommend ermittelt wurde, sind auch alle seine Teilmengen häufig und ihr Support ist bekannt. Jetzt wird die Konfidenz der entstandenen Regeln berechnet. In Abhängigkeit von *minconf* werden die Regeln nun akzeptiert oder aufgegeben. Die Menge aller akzeptierten Regeln bildet die Menge der gefundenen Assoziationsregeln.

Weitere Assoziationsalgorithmen: AIS (siehe [Mue95]) ist einer der ersten Algorithmen, die entwickelt wurden, um Assoziationregeln zu generieren. Ein Nachteil von AIS ist jedoch, dass er zu viele mögliche Kandidaten betrachtet. Ein weiteres Problem von AIS und auch von Apriori ist, dass bei jedem Durchlauf die gesamte Datenbank gelesen wird, obwohl viele Itemmengen in späteren Durchläufen nicht mehr gebraucht werden. Der AprioriTid Algorithmus (siehe [FPSSU96]) löst dieses Problem, indem er die Datensätze einmal liest und die Daten für spätere Durchläufe zwischenspeichert. Da er die Kopie im Hauptspeicher hält, ist die Größe der Datenmenge begrenzt, für die der Algorithmus arbeitet. Der PARTITION Algorithmus (siehe [Mue95]) nutzt eine ähnliche Repräsentation der Daten in Form von TID-Listen (intersecting sets) wie AprioriTid. Zusätzlich verwendet er

eine Partitionierungstechnik, die es ermöglicht, Datenportionen im Hauptspeicher zu verarbeiten, so dass größere Datenmengen als bei ArioriTid betrachtet werden können.
Der AprioriHybrid Algorithmus ist eine Kombination von Apriori und AprioriTid, wobei für die ersten Durchläufe Apriori angewandt wird und wenn die in Frage kommenden Daten in den Hauptspeicher passen, wird zu AprioriTid gewechselt.
DIC, eine Weiterentwicklung von Apriori, versucht, wie auch AprioriTid, die Anzahl der Datenbankdurchläufe zu minimieren. Die Idee dabei ist jedoch, die strikte Trennung zwischen dem Erzeugen und dem Zählen der Kandidaten zu lockern (siehe [HGN00]).
Andere Ansätze (siehe [KMR$^+$94]) erlangen durch die Vorgabe von Regelmustern (rule templates), eine Einschränkung des Suchraumes für Assoziationsregeln. Diese Regelmuster beschreiben die Struktur derjenigen Regeln, die als Analyseergebnis von besonderem Interesse sind.

2.3 Clustering

Die Aufgabe des Clustering besteht darin, eine Unterteilung der Instanzen in sinnvolle Klassen vorzunehmen. Es wird nach einer Beschreibung dieser Klassen gesucht. Clustering kann auch als unüberwachte Klassifizierung bezeichnet werden. Im Gegensatz zur eigentlichen Klassifikation, wo gelernt wird, wodurch sich vorgegebene Klassen auszeichnen, geht es bei der unüberwachten Klassifikation um das Identifizieren statistisch signifikanter Klassen innerhalb der Daten. In [GRS98] wird Clustering definiert als ein Problem, bei dem n Datenpunkte (Instanzen) in einem d-dimensionalen metrischen Raum gegeben sind. Diese Datenpunkte sollen in k Cluster unterteilt werden, so dass die Datenpunkte innerhalb eines Clusters zueinander ähnlicher sind als zu Datenpunkte in anderen Clustern. Die Dimension des Raumes ist von der Anzahl zu betrachtenden Attribute abhängig.
Es wurden verschiedene Cluster-Algorithmen entwickelt. Im allgemeinen werden die Klassen so generiert, dass jede Instanz nur einer Klasse angehört. Einige Algorithmen erlauben es, dass Instanzen mehreren Clustern angehören können, d.h. die Cluster überlappen sich in diesem Fall. Andere wiederum erzeugen eine hierarchische Cluster-Struktur, die eine Folge von Partitionen darstellt, in der jede Partition in der nächsten Partition der Folge enthalten ist. Ein anderer Ansatz ordnet den Instanzen Wahrscheinlichkeiten zu, die angeben, wie wahrscheinlich es ist, dass eine Instanz einer bestimmten Klasse angehört. [RF00] stellt fest, dass sich die bisher entwickelten Cluster-Algorithmen im wesentlichen in drei Punkten unterscheiden:

1. in der Art und Weise, wie die Ähnlichkeit zwischen Instanzen definiert wird
2. in der Funktion, die genutzt wird, um die Relevanz einer Partition zu bewerten
3. in der Art der Suchoptimierung, denn eine erschöpfende Suche über alle Partitionen ist in der Regel zu aufwendig

Im folgenden werden zwei Cluster-Algorithmen beschrieben.

COBWEB: Das COBWEB System von Fisher (siehe [HK00a]) implementiert einen inkrementell arbeitenden Cluster-Algorithmus, der in jeder Phase einen Baum generiert. Das heißt, jede neue unklassifizierte Instanz wird in eine Hierarchie eingeordnet und verändert diese dabei. Verschiedene Operationen dienen der Aktualisierung des Baumes, z.B. kann ein Blatt neu erzeugt werden oder ein Knoten kann aufgespalten werden bzw. mit einem benachbarten Knoten verschmelzen. Bei der Entscheidung, wie und wann eine Aktualisierung durchgeführt wird, spielt ein globales Qualitätskriterium eine signifikante Rolle. Jede neue Instanz wird zunächst jedem vorhandenen Blatt einzeln zugeordnet. Das Qualitätsmaß wird jedesmal berechnet. Danach werden die Werte verglichen, um festzustellen, welchem Blatt die Instanz am besten zugeordnet werden sollte. Wie sich das Qualitätsmaß berechnet, kann in [WF01] nachgelesen werden, dort wird das Maß als Kategorienützlichkeit bezeichnet. Während COBWEB nur mit nominalen Attributen arbeitet, führt CLASSIT (siehe [Car90]) die gleichen Verschmelzungs- und Aufteilungsoperationen für numerische Attribute durch.

AutoClass: AutoClass (Automatic Classification) ist ein Bayessches Clustering-Verfahren, das nach der besten posterior Wahrscheinlichkeitsklassifikation sucht. Die Eingabe umfasst die Daten mit nominalen oder numerischen Attributen und ein Klassenmodell (siehe [FPSSU96]). Das Verfahren kann mit fehlenden Werten umgehen und es erlaubt sogar Korrelationen zwischen den Attributen innerhalb einer Klasse. Diese Attribute werden dann mit der gleichen Wahrscheinlichkeitsverteilung modelliert. AutoClass findet eine Menge von Klassen, die die maximale Wahrscheinlichkeit hinsichtlich der Daten und dem Modell aufweist. Die Anzahl der zu findenden Klassen wird automatisch bestimmt und muss nicht vom Nutzer vorgegeben werden. Ausgegeben werden eine Menge von Klassenbeschreibungen. Da Instanzen in mehreren Klassen enthalten sein können, werden auch Informationen zur jeweiligen Klassenzugehörigkeit angegeben.

Weitere Cluster-Algorithmen: Ein weiterer Cluster-Algorithmus ist BIRCH (siehe [HK00a]). Er arbeitet in zwei Phasen. Als erstes führt er ein Preclustering durch, wobei dicht besetzte Regionen von Datenpunkten zusammengefasst werden, um die Datenmenge zu reduzieren. Dann wird das eigentliche Clustering auf den reduzierten Daten durchgeführt und eine hierarchische Cluster-Struktur erstellt. Ein anderer Algorithmus namens CURE (Clustering Using Representatives) zieht anstelle des Preclustering eine zufällige Stichprobe, um die Daten zu reduzieren (siehe [GRS98]). Außerdem ist CURE robuster gegenüber Ausreißern in den Daten.

2.4 Zusammenfassung

In diesem Kapitel wurden verschiedene Verfahren des Data Mining vorgestellt. Zunächst wurde jedoch definiert, was in dieser Arbeit unter Data Mining verstanden wird. Data Mining, als Teilschritt des KDD-Prozesses, versucht mit Hilfe von Methoden des Maschinellen Lernens, der Mustererkennung und der Statistik Prognosen aufzustellen bzw. Beschreibungen zu finden. Des weiteren wurde eine Übersicht erarbeitet, die zunächst die gebräuchlichen Lernverfahren des Data Mining zusammenfasst und die dann zwei Arten von Verfahren zeigt, nämlich die des überwachten Lernens und die des unüberwachten Lernens. Außerdem wurden drei Verfahrensfamilien unterschieden.
Die Klassifikation generiert ein Klassifikationsmodell, um neue Datensätze einer bestimmten Klasse zuordnen zu können bzw. um unbekannte Attributwerte vorherzusagen. Dieses Modell kann in Form von verschiedenartigsten Regelmengen oder Bäumen erzeugt werden, wie an der Vielfalt der vorgestellten konkreten Algorithmen erkennbar ist.
Des weiteren wurde die Assoziation als Verfahrensfamilie vorgestellt. Sie versucht, Regelmäßigkeiten in Daten zu erkennen. Zwei notwendige Schritte zum Aufstellen der Assoziationsregeln wurden genannt und am Beispiel des Apriori Algorithmus' genauer beschrieben.
Als dritte und letzte Verfahrensfamilie wurde das Clustering beschrieben. Verfahren des Clustering versuchen Beschreibungen zu statistisch relevanten Klassen zu finden. Zwei unterschiedlich arbeitende Verfahren wurden erläutert. Das eine, COBWEB, ist ein inkrementelles Verfahren, das Instanzen in einen Baum einordnet, um eine hierarchische Cluster-Struktur aufzubauen. Das andere, AutoClass, ist ein Verfahren, das nach der besten posterior Wahrscheinlichkeitsklassifikation sucht.
Abschließend kann bemerkt werden, dass sich jeder Data-Mining-Algorithmus durch drei Hauptkomponenten (siehe [FPSS96]) auszeichnet:

- die Modellrepräsentation, die Sprache, die genutzt wird, um die zu entdeckenden Muster zu beschreiben (Baum, Regeln, ..)
- die Modellevaluierungskriterien/Fitnessfunktionen, die angeben, wie gut ein bestimmtes Muster den Zielen des KDD-Prozesses entsprechen (Qualitätsmaß, Konfidenz, ..), und Strategien zur Vermeidung einer eventuellen Überanpassung an Trainingsdaten (Pruning)
- die Suchmethode, die sich aus der Parametersuche und der Modellsuche zusammensetzt und versucht die Evaluierungskriterien bestmöglichst zu erfüllen (Tiefensuche, Breitensuche, ..)

Durch die Auswahl der Komponenten wird der Bias eines Algorithmus' induziert. Der Bias bestimmt das Lernverfahren. Dementsprechend lassen sich auch Bewertungskriterien ableiten, die genutzt werden, um einen geeigneten Algorithmus für

eine Data-Mining-Aufgabe zu bestimmen. Ausschlaggebend sind dabei vor allem die Genauigkeit, mit der etwas mit einem Verfahren vorhergesagt oder beschrieben wird, sowie Trainings- und Testzeiten, die das Verfahren benötigt, um ein Modell zu generieren. In der Literatur werden ein oder mehrerer derartiger Angaben auch als Leistung bzw. Performanz des Verfahrens zusammengefasst. In den nächsten Kapiteln geht es nun darum, wie die Auswahl eines Data-Mining-Verfahrens automatisiert werden kann, so dass derjenige Algorithmus mit der besten Performanz für eine gegebene Aufgabe bestimmt wird.

3 Meta-Lerner

Um eine Data-Mining-Aufgabe bearbeiten zu können, muss aus einer Vielzahl von Data-Mining-Techniken, siehe Kapitel 2, ein Verfahren ausgewählt werden. Das Problem liegt nun darin, den geeignetsten Algorithmus für die gegebene Aufgabe herauszufinden, denn nach der Veröffentlichung von Wolpert's sogenannten "No Free Lunch Theorems for Search" (NFL-Theorem) wird es allgemein akzeptiert, dass es keinen Algorithmus gibt, der andere Algorithmen in allen Lernproblemen dominiert (siehe [WM95]). Die Performanz der Verfahren wird vor allem an der Genauigkeit gemessen, mit der sie etwas vorhersagen bzw. beschreiben, aber auch Trainings- und Testzeiten sind relevant für die Auswahl. Die Suche nach einem geeigneten Verfahren wird als Modellauswahl-Problem (model selection problem) bezeichnet. Dieses Problem umfasst nicht nur die Suche nach einem geeigneten Algorithmus, sondern auch das Festlegen seiner Parameter.
Meta-Lerner versuchen, das Modellauswahl-Problem mit Hilfe von Verfahren des Maschinellen Lernens zu lösen. Die Idee dabei ist, die Ergebnisse von zuvor bearbeiteten Aufgaben zu nutzen. Das Erstellen einer Art Kategorisierung von Aufgaben in Typen ist erforderlich zusammen mit einer Vorschrift, die aussagt, dass ein bestimmter Aufgabentyp gut von einem bestimmten Algorithmus bearbeitet wird. Es soll also eine Zuordnung von Aufgaben zu Methoden gefunden werden, so dass die Suche nach der richtigen Methode für eine neue Aufgabe nicht immer wieder von neuem angefangen muss und so Zeit gespart wird.
[CS97] gibt eine allgemeine Definition für das Meta-Lernen. Es wird beschrieben als das Lernen von Meta-Wissen über bereits gelerntes Wissen. Der Aufbau eines Meta-Lerners entspricht etwa der Meta-Level Architektur, die Maes in [MN88] beschreibt. Die Architektur umfasst zwei Ebenen, die Objekt-Ebene und die Meta-Ebene. Die Aufgabe der Meta-Ebene (Meta-Lerner) besteht dabei in der Lieferung von Informationen über die Berechnungen der Objekt-Ebene (Berechnungen der Basislerner von denen einer ausgewählt werden soll). Diese Meta-Informationen wiederum beeinflussen spätere Berechnungen der Objekt-Ebene. Der Umfang und die Form der zu berechnenden Meta-Informationen sind vom Ziel des Meta-Lerners abhängig. Ziele des Meta-Lernens im Data-Mining-Bereich werden im folgenden Abschnitt vorgestellt.

3.1 Ziele des Meta-Lernens im Data-Mining-Bereich

Die Ziele, die mit einem Meta-Lerner verfolgt werden, sind unterschiedlich. In 2.1 wurde bereits angesprochen, dass zur Kombination mehrerer Modelle unterschiedlicher Klassifikationsverfahren ein Metalernsystem eingesetzt wird. Die als Stacking bezeichnete Methode versucht dadurch, eine **optimale Kombination von Modellen** herauszufinden. Ziel ist es, eine genauere Konzeptbeschreibung zu finden, als es mit den einzelnen Basisklassifikatoren möglich ist. Außerdem kann ein solches Multistrategiesystem einen viel größeren Bereich von Klassifikationspro-

blemen lösen, da sie die Vorzüge der gegenseitigen Ergänzung nutzen können.
Für [CS97] dient Meta-Lernen dem **"Zusammenkleben" mehrerer Wissensquellen**. Zunächst werden Teilmengen der Gesamtdatenmenge, die z.B. verteit in einem Netzwerk vorliegen, parallel durch separate Lernprozesse analysiert. Danach werden die einzelnen Ergebnisse durch einen Meta-Lerner zusammengefügt. Dies soll ein effizienteres Data Mining für große Datenmengen ermöglichen.
Ein weiteres Ziel ist das Auswählen von Lernverfahren. Die **Lernverfahrenauswahl** stellt im Gegensatz zur Verfahrenskombination die Frage nach dem Entweder-Oder. Hierbei gilt es, das geeignetste Verfahren für eine Data-Mining-Aufgabe herauszufinden. Es existieren auch Ansätze, siehe [SB00], [BS00] und [BPK], die anstelle der Auswahl eines einzelnen bzw. einer kleinen Gruppe von Algorithmen versuchen, eine Rangfolge der gegebenen Algorithmen zu erstellen, die aussagt, welche Algorithmen wie gut zur Lösung einer Aufgabe geeignet sind und um wieviel besser bestimmte Algorithmen im Vergleich zu anderen sind.

3.2 Lernverfahrenauswahl durch Meta-Lerner

In dieser Diplomarbeit soll mit Hilfe eines Meta-Lerners ein Data-Mining-Verfahren für die Lösung einer Data-Mining-Aufgabe aus einer Menge von Verfahren ausgewählt werden. Bisher werden die Lernverfahren durch einen Experten ausgewählt, der weiß, welche Lernverfahren bei welchen Aufgaben gute Ergebnisse liefern. Dabei werden oft bereits bearbeitete Aufgaben mit der aktuellen verglichen. Wenn es das Ziel ist, das Auswahlverfahren zu automatisieren, dann gibt es laut [Ben99] zwei mögliche Alternativen. Auf der einen Seite kann man das Wissen der Experten extrahieren und ein regelbasiertes Auswahlsystem entwickeln. Jedoch ist diese Möglichkeit stark vom Wissen der Experten abhängig und davon, wie Systeme dieses Wissen verwenden können. Auf der anderen Seite kann man ein Lernsystem dazu bringen, Informationen über früher bearbeitete Aufgaben zu speichern und Ähnlichkeiten zur aktuellen herauszufinden. In diesem Fall lernt das System Vorzüge zu bestimmten Lernverfahren mit Problemstellungen in Verbindung zu bringen, d.h. es lernt, wie es eine gegebene Aufgabe am besten lernt.
Wie gut ein solches System derartige Vorzüge erlernen kann, ist von drei wesentlichen Faktoren abhängig:

- Zum einen ist natürlich das Lernverfahren entscheidend, das dem System zugrunde liegt. Das Modellauswahl-Problem kann auch als überwachtes Klassifikationsproblem bezeichnet werden, wobei sich die zu lernenden Klassen aus den zur Verfügung stehenden Algorithmen ergeben, z.B. Regellerner, Entscheidungsbaum, IBL (Instance Based Learner) oder ILP. Jeder Klassifikationsalgorithmus, siehe Kapitel 2.1, kann demzufolge als Meta-Lerner bzw. Meta-Klassifizierer eingesetzt werden, auch das Einsetzen des Fallbasierten Schließens (Case-Based Reasoning, CBR) ist möglich. Beispiele für den Einsatz von Meta-Klassifizierern und dem Fallbasiertem Schließen werden in 3.7 beschrieben.

- Zum anderen hängt der Lernerfolg von den Daten ab, die dem System zur Verfügung stehen und aus denen das System die Konzeptbeschreibung ableitet. Im Fall des Modellauswahl-Problems wird eine Konzeptbeschreibung aus Metadaten abgeleitet. Durch diese Metadaten werden die Problemstellungen, die das System bereits bearbeitet hat, beschrieben. Auch die neue Aufgabe wird auf diese Art dargestellt, um einen entsprechenden Vergleich der Aufgaben zu ermöglichen.

- Ausschlaggebend für den Erfolg eines Meta-Lerners ist außerdem, wie stark sich die Anwendungsbereiche der potentiell geeigneten Methoden, die dem Meta-Lerner zur Auswahl stehen, unterscheiden. Ähneln sich die Verfahren sehr in ihren Ergebnissen, so ist das Finden von Vorzügen zu bestimmten Verfahren für gegebene Problemstellungen erschwert.

Der Name Meta-Lerner rührt nicht nur allein daher, dass die dem System zur Verfügung stehenden Daten Metadaten sind, sondern weil im Prinzip Data Mining über Data Mining, also Meta Data Mining, durchgeführt wird. Da Data-Mining-Verfahren Lernverfahren sind, siehe Kapitel 2, spricht man von Meta-Lernern. So wird beispielsweise auch die Kommunikation über die Kommunikation als Meta-Kommunikation bezeichnet.

Meta-Lernen zur Lernverfahrenauswahl ist eine induktive Aufgabe. Abbildung 2 zeigt dazu ein kleines Beipiel. Es sind drei bereits bearbeitete Aufgaben gegeben. Für jede dieser Aufgaben ist bekannt, welches das geeignetste Verfahren ist. Das jeweilige Verfahren wurde den Aufgaben als Klasse (Label) zugeordnet. Mit Hilfe dieser Beispielaufgaben soll nun eine Lösung, das geeignetste Verfahren, für die neue Aufgabe gefunden werden.

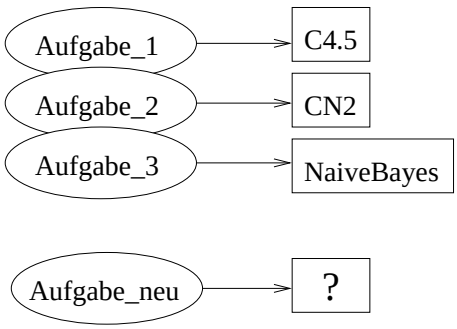


Abbildung 2: Lernverfahrenauswahl als induktive Aufgabe

Die Aufgabe des Meta-Lerners ist es, derartige Beispielaufgaben zu nutzen, um zu lernen, wie neue Aufgaben entsprechend einer geeigneten Strategie richtig klassifiziert werden.

Die Frage, wie das geeignetste Verfahren für eine Aufgabe bestimmt werden kann, wird im nächsten Abschnitt beantwortet.

3.3 Bestimmung des geeignetsten Lernverfahrens

Die Bestimmung des geeignetsten Lernverfahrens ist offensichtlich ein elementarer Schritt, wenn es darum geht, eine Zuordnung zwischen Aufgabenstellung und bestem Verfahren zu finden. Um die Vorteile bzw. Schwächen bestimmter Verfahren herauszufinden, bedarf es Möglichkeiten Algorithmen miteinander zu vergleichen. [BP] unterscheidet generell zwischen theoretischen und empirischen Vergleichsmöglichkeiten. Jedoch geht es hier nicht um das allgemeine Vergleichen von Algorithmen, sondern speziell um das Vergleichen von Ergebnissen, die durch Bearbeiten einzelner Aufgaben erzielt wurden.
Um für ein Trainingsbeispiel des Meta-Lerners das geeignetste Lernverfahren zu bestimmen, muss also eine Evaluierungsstrategie angewandt werden, die solche Metriken verwendet, die aus den einzelnen Ergebnissen der Algorithmen bestimmt werden können. Das Ergebnis einer Evaluierung bestimmt die Klasse bzw. das Label des Trainingsbeispiels. Da bereits Evaluierungsstrategien speziell für Klassifikationsaufgaben entwickelt wurden, werden diese im folgenden vorgestellt.
Am häufigsten wird die Fehlerrate (bzw. die Erfolgsrate) als Metrik für eine Evaluierungsstrategie genutzt. Die Aufgabe eines Klassifizierers besteht darin, die Klasse von Instanzen vorherzusagen. Bestimmt er die richtige Klasse, wird das als Erfolg gewertet, andernfalls als Fehler. Die Fehlerrate kann auf unterschiedliche Weise bestimmt werden:

- **mit Hilfe von Testdaten:** Hier wird der Klassifizierer einmalig mit Trainingsdaten trainiert und anschließend mit Testdaten getestet. Dies erfordert eine entsprechende Menge an Daten, die in zwei unabhängige Teilmengen (Trainings- und Testdaten) aufgeteilt werden kann und zwar so, dass sowohl die Trainings- als auch die Testdaten repräsentative Beispiele für das zugrundeliegende Problem darstellen. Ungefähr zwei Drittel der Daten werden zum Trainieren, also zum Aufbau des Klassifizierers verwendet. Das andere Drittel der Daten wird auf den generierten Klassifizierer angewandt. Das mit Hilfe der Testdaten ermittelte Verhältnis der Falsch-Klassifizierungen zur Gesamtanzahl der klassifizierten Instanzen ergibt die Fehlerrate des Klassifizierers.
  Stehen nicht ausreichend Daten zur Verfügung, so werden andere Methoden, wie die folgenden zwei, angewandt.

- **durch Kreuzvalidierung:** Bei der Kreuzvalidierung werden die zur Verfügung stehenden Daten zunächst in Partitionen aufgeteilt. Jede dieser Partitionen wird nacheinander für das Testen verwendet (Holdout), während der Rest für das Training verwendet wird. Wenn die Wahl der Partitionen so

gewählt wird, dass in jedem Fall jede Klasse in der Trainings- und Testmenge repräsentiert wird, so bezeichnet man das als Stratifikation. Standardmethode zur Ermittlung der Fehlerrate ist laut [WF01] die stratefizierte zehnfache Kreuzvalidierung. Das heißt, zehnmal wird eine von zehn Partitionen als Testmenge gewählt, um die Fehlerrate des durch die restlichen neun Partitionen erzeugten Klassifizierers zu ermitteln. Die Gesamtfehlerrate ergibt sich, wenn man die zehn Fehlerraten mittelt.

- **durch Bootstrap:** Im Gegensatz zur Kreuzvalidierung wird die Datenmenge nicht in disjunkte Teilmengen partitioniert, sondern es wird eine Stichprobe mit Ersetzen entnommen. Ist ein Datensatz mit n Instanzen gegeben, dann wird n mal gezogen, um die Trainingsmenge festzulegen. Nicht gezogene Instanzen bilden die Testmenge. Das Ergebnis eines Durchganges wird aus der Fehlerrate und dem Resubstituierungsfehler (siehe [WF01]) ermittelt. Die Bootstrap-Prozedur wird mehrmals wiederholt, mit unterschiedlichen Stichproben als Trainingsmenge, und die Ergebnisse werden gemittelt, um die Gesamtfehlerrate zu bestimmen.

In [MST94] wurden die Lernverfahren mit einer niedrigen Fehlerrate als anwendbar (applicable), alle übrigen Verfahren als nicht anwendbar (inapplicable) für eine gegebene Aufgabe gekennzeichnet. Dem Nutzer wurde meistens eine kleine Menge von Verfahren empfohlen. Um jedoch auch Aussagen darüber zu treffen, um wieviel besser ein Verfahren gegenüber anderen ist, entwickelte man verschiedene Ranking-Methoden, wobei jeweils das Verfahren mit dem höchsten Rankwert das geeignetste Lernverfahren darstellt. [BS00] untersucht und vergleicht unterschiedliche Ranking-Methoden. So z.B. die Methode Success Rate Ratios (SRR), die die Algorithmen entsprechend ihrer Vor-/Nachteile gegenüber anderen Algorithmen rankt oder auch die Methode Significant Wins (SW), die die Algorithmen paarweise miteinander vergleicht.
Das ausschließliche Betrachten der Fehlerrate wird jedoch als nachteilig empfunden, da es nur ein einzelnes Maß ist, das noch nicht genügend Aufschluss über die Verfahren gibt. Deshalb wurden Evaluierungsstrategien entwickelt, wie sie in [SCB] und [SB00] beschrieben werden. Sie lassen zusätzlich die Zeit einfließen, die das Lernverfahren zum Trainieren und Testen eines Modelles benötigt. Ein weiterer Vorteil dieser Methoden ist, dass sie dem Nutzer die Möglichkeit bieten, Einfluss auf die Auswahl nehmen zu können. So kann der Nutzer im Fall von [SB00] bestimmen, wie stark die Zeit gegenüber der Erfolgsrate, dem Gegenstück zur Fehlerrate, gewichtet wird. In [SCB] kann der Nutzer Grenzen angeben, z.B. eine bestimmte Erfolgsrate, die die in Frage kommenden Verfahren nicht unterschreiten dürfen. [SCB] beschreibt eine einfache Evaluierungsstrategie, die für weitere Eigenschaften erweiterbar ist, z.B. den benötigten Speicherplatzbedarf eines Verfahrens zur Ausführung von Data-Mining-Aufgaben. Im Abschnitt 3.6 werden weitere Kriterien genannt, die auch auf die Bestimmung des geeignetsten Lernverfahren Einfluss nehmen können.

Im anschließenden Abschnitt werden unterschiedliche Möglichkeiten aufgeführt, die zur Beschreibung von Data-Mining-Aufgaben genutzt werden können.

3.4 Aufgabenbeschreibung durch Metadaten

Die Aufgabenbeschreibung (task characterisation) spielt eine entscheidende Rolle in den bisherigen Ansätzen des Meta-Lernens zur Lernverfahrenauswahl. Sie ermöglicht dem Meta-Lerner eine Abbildung zu finden, um von einer bestimmten Beschreibung auf ein passendes Modell schließen zu können. [BGCK00] stellt fest, dass es bisher drei unterschiedliche Ansätze zur Aufgabenbeschreibung gibt: eine statistisch und informationstheoretisch basierte Aufgabenbeschreibung, das Landmarking und eine Entscheidungsbaum-basierte Beschreibung. Diese Ansätze wurden bisher auf Klassifikationsaufgaben beschränkt, wobei nicht auszuschließen ist, dass sie auch zur Beschreibung anderer Aufgaben dienen können. Auf diese drei Arten wird im folgenden näher eingegangen.

3.4.1 statistische und informationstheoretische Aufgabenbeschreibung

Dieser Ansatz beschreibt eine Problemstellung mit Hilfe von statistischen und informationstheoretischen Kennzahlen. Diese werden auf der Basis der zu analysierenden Daten berechnet. Im Rahmen des StatLog Projektes[1] beschäftigte man sich bereits mit der Frage, warum bestimmte Algorithmen einige Daten gut und andere weniger gut verarbeiten können. Um dieser Frage nachzugehen, versuchte man, Klassifikationsaufgaben durch drei Gruppen von Kennzahlen zu beschreiben, die hier genannt seien:

1. Die erste Gruppe von Kennzahlen werden als einfache Maße bezeichnet. Sie sollen einen ersten Eindruck von Komplexität oder der Größe des Problems geben. Zu diesen Maßen gehören die Gesamtanzahl von Datensätzen und Attributen, die Anzahl von Klassen und speziell die Anzahl von binären Attributen.

2. Die zweite Gruppe bilden die statistischen Maße. Sie charakterisieren die kontinuierlichen Attribute der Daten. Zu ihnen gehören Angaben über die Verteilung von Attributen und ihren Abhängigkeiten untereinander, wie z.B. der mittlere absolute Korrelationskoeffizient.

3. Die dritte Gruppe der informationstheoretischen Maße beschreibt die diskreten bzw. kategorischen Attribute. Zu dieser Gruppe gehören Kennzahlen, wie z.B. die Entropie der Attribute und der Klassen, die Verrauschung von Attributen und die Angaben über irrelevante Attribute.

[1] ESPRIT Projekt 5170.

Nähere Erläuterungen zu den einzelnen Kennzahlen können in [MST94] nachgelesen werden. Die Maße wurden genutzt, um mit Hilfe des C4.5 Klassifikationsalgorithmus, siehe Kapitel 2.1, die Leistung von Klassifikationsalgorithmen einschließlich des C4.5 vorherzusagen. Ein Kritikpunkt war jedoch, dass einige statistische Maße zu komplex sind und demzufolge einen hohen Berechnungsaufwand erforderten, der sogar den des Ausführens mancher Lernalgorithmen überstieg. In [PBGC00] wird berichtet, dass einige dieser Tests eine Komplexität von $O(n^3)$ erreichten, wobei n die Anzahl der Datensätze ist. Dem Bericht zufolge scheint eine Komplexität von $O(n\,log\,n)$ für angemessener. Da eine vollständige Beschreibung der Daten durch statistische Maße schnell in ihrer Komplexität steigt, stellt sich die Frage, welche der Statistiken genutzt werden sollten. [Ben99] bezweifelt jedoch, dass ein statistikbasiertes Meta-Lernsystem eine schnelle Antwort auf diese Frage liefert. Vorteilhaft ist es an dieser Stelle sicherlich, wenn die zu analysierenden Daten in einer Datenbank vorliegen. So könnte auf bereits von der Datenbank ermittelte Statistiken zurückgegriffen werden, um den Berechnungsaufwand zu verringern. In dem Datenbanksystem DB2 von IBM werden unter anderem Statistiken wie die Anzahl unterschiedlicher Werte eines Attributes sowie Angaben über ihre Verteilung mitgeführt (siehe [Cha98]).
[SB00] nutzt in seinem Ansatz "Zoomed Ranking" das von Lindner und Engels entwickelte Data Characterization Tool (DCT) (siehe [LS99]). Das DCT berechnet, wie es in StatLog der Fall ist, informationstheoretische und statistische Informationen aus den Daten. Dadurch können im "Zooming"-Schritt die gegebenen Daten analysiert und relevante, ähnliche Daten aus der Menge der bereits bearbeiteten gefunden werden. Es wird jedoch bemerkt, dass diese Art der Meta-Attribute nur gewählt wurde, weil sie durch das DCT zur Verfügung gestellt wird.

3.4.2 Landmarking

Ein interessanter neuer Ansatz zur Aufgabenbeschreibung ist Landmarking. Beim Landmarking wird die Leistung von einigen einfachen und effizienten Algorithmen zur Bearbeitung von Aufgaben bestimmt. Diese Algorithmen bezeichnet man als Landmark-Lerner bzw. Landmarker. Ihre Performanzwerte sind die Metadaten der Aufgabe, d.h. jede Aufgabe wird auf Meta-Ebene durch einen Vektor von Landmarkerperformanzwerten beschrieben. Die Dimensionalität ergibt sich aus der Anzahl der Landmarker.
Für die Trainingsaufgaben wird nun, wie auch bei anderen Ansätzen, durch Ausprobieren der jeweils beste Lernalgorithmus gesucht und außerdem die Performanz der Landmarker bestimmt. Auf diese Art wird eine Beziehung zwischen Metadaten, also den Performanzwerten der Landmarker, und den jeweils geeignetsten Algorithmen hergestellt, die dann durch den Meta-Lerner gelernt werden kann. Die neue Aufgabe wird zunächst von den Landmarkern ausgeführt, d.h. die Metadaten werden ermittelt. Die Metadaten wiederum führen mit Hilfe des vom Meta-Lerner entwickelten Modells zur Auswahl des geeignetsten Lernverfahren.

Abbildung 3 zeigt ein Beispiel, wie die Performanz zweier Landmarker L_1 und L_2

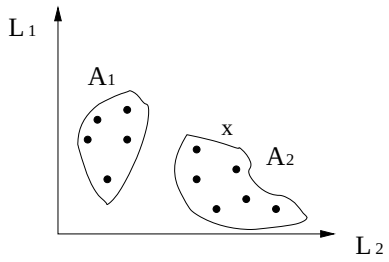


Abbildung 3: Landmarking

genutzt wird, um entweder Algorithmus A_1 oder A_2 als geeignetsten komplexen Algorithmus auszuwählen. Die Punkte entsprechen bereits bearbeiteten Aufgaben, die mit einem Label versehen wurden. Das Label entspricht dem geeignetem Algorithmus. In der Abbildung wurden die Aufgaben mit gleichem Label zu einer Punktwolke zusammengefasst. Die zu klassifizierende Aufgabe wird durch das x dargestellt. In diesem Fall ist es offensichtlich, dass der Meta-Lerner A_2 als geeigneter Algorithmus empfohlen wird. Es sei angemerkt, dass die Abbildung stark vereinfacht ist, so wurden nur zwei Landmarker und nicht überlappende Punktmengen dargestellt. In Experimenten werden häufig mehrere Landmarker verwendet, deren Performanzwerte auch zu Überschneidungen der Punktmengen führen.
Des weiteren spielt die "Area of Expertise" eine wichtige Rolle. Sie umfasst diejenige Klasse von Aufgaben, die ein Lerner gemessen an einem Performanzmaß, z.B. der Vorhersagegenauigkeit, gut löst. Die beiden Punktwolken in Abbildung 3 entsprechen den "Areas of Expertise" von A_1 bzw. A_2. Eine Aufgabe kann demzufolge durch die Menge der "Areas of Expertise", denen sie angehört, beschrieben werden. Landmarking versucht nun, die Lage eines bestimmten Lernproblems im Raum aller Lernprobleme zu ermitteln. Entsprechend seiner Position kann dann ein komplexerer Algorithmus zur Bearbeitung des Lernproblems bestimmt werden.
In [BGC00a] wird Landmarking mit dem Ansatz der Kreuzvalidierung zur Modellauswahl verglichen. Anstelle des Ausführens der komplexen Lernalgorithmen mit einem Teil der Daten, lässt Landmarking einfache Verfahren über die gesamten Daten laufen.
Ein Nachteil dieses intuitiven Ansatzes ist, dass die Auswahl der Landmarker nicht trivial ist:

- Es ist klar, dass nur Algorithmen, die dem betreffenden System zu Verfügung stehen, als Landmarker fungieren können, d.h. sie müssen im sogenannten Lerner-Pool des Systems enthalten sein.

- Des weiteren werden Landmarker mit einer geringen Zeitkomplexität benötigt, d.h. die Laufzeitkosten der gewählten Landmarker sollten geringer sein

als die der komplexeren Lernverfahren. Andernfalls wäre es besser, alle Lerner auszuprobieren, um den besten herauszufinden.

- Außerdem muss die Anzahl der Landmarker so gewählt werden, dass die Aufgaben ausreichend gut beschrieben sind. Die Anzahl der Meta-Attribute richtet sich nach der Anzahl der Landmarker, ihre Werte entsprechen den Performanzwerten der Landmark-Lerner.

- Landmarker sollten sich in ihren "Areas of Expertise" ausreichend unterscheiden. Wenn es keinen signifikanten Unterschied zwischen zwei Algorithmen gibt, dann werden sie als "tied" bezeichnet. In diesem Fall ist es egal, welcher Algorithmus gewählt wird. Die Nähe der "Area of Expertise" zweier Algorithmen lässt auf Ähnlichkeiten ihrer Verfahren schließen (siehe [BGC00a] und [BGC00c]).

Es wurden bereits einige Experimente mit Klassifikationsalgorithmen durchgeführt. In [BGC00a] und auch in [BGCP$^+$00] beschreibt man Experimente, die unterschiedliche Zusammensetzungen von Landmarkern untersuchen. Das langfristige Ziel derartiger Experimente ist, eine universelle Menge von Landmarkern zu finden, die in jedem Lerner-Pool genutzt werden kann. Hier wurde bereits herausgefunden, dass Naive Bayes als Landmarker keinen großen Einfluss auf das Meta-Lernen hat, das Ergebnis verbessert sich in einigen Fällen sogar, wenn Naive Bayes aus der Menge der Landmarker entfernt wird. Erklärt wird dies in [BGC00a] wie folgt: "This can be explained by the attribute independence assumption of Naive Bayes. We can conjecture that its attribute independence assumption is not appropriate for meta-learning with landmarking."
In einigen Experimenten, siehe [BGC00b] und [BGC00a], fungieren Landmarker gleichzeitig auch als Lerner, der zur Bearbeitung der Aufgabe ausgewählt werden kann. Dies gilt als unbedenklich, denn der Meta-Lerner kennt die Identität der Landmarker nicht.
In [BGC00b] wird Landmarking mit dem statistischen Ansatz verglichen, wobei Landmarking meistens niedrigere Fehlerraten erzielte. Das Anreichern der Landmarker-Performanzwerte mit informationstheoretischen Kennzahlen, also die Kombination beider Ansätze, beeinträchtigt nur die Ergebnisse des nativen Landmarking. Nach einer Erklärung dafür wurde leider nicht gesucht, man könnte es aber, auf die unglückliche Auswahl der sechs betrachteten informationstheoretischen Kennzahlen zurückführen. Insgesamt kommt man zu dem Schluss, dass Landmarking durchaus zum Meta-Lernen eingesetzt werden kann und mit dem statistischen Ansatz konkurrieren kann. Dies wird durch Ergebnisse, die in [BGC00c] veröffentlicht wurden, bestätigt. Dort hat man die beste Wahl eines Algorithmus' für eine Aufgabe mit der Auswahl des Landmarking verglichen.
Während beim Landmarking die Performanzwerte eines Entscheidungsbaumverfahren als Metadaten von Interesse sein können, steht beim nächsten Ansatz die eigentliche Struktur des Baumes zur Ermittlung von Metadaten im Vordergrund.

3.4.3 Entscheidungsbaum-basierte Aufgabenbeschreibung

Es gibt zwei unterschiedliche Ansätze, die Entscheidungsbäume zur Beschreibung von Aufgaben nutzen. Der Grundgedanke ist bei beiden ähnlich und zwar, dass die Gestalt eines Entscheidungsbaum, der aus den zu analysierenden Daten generiert wurde, Eigenschaften besitzt, die stark von der Aufgabe und den gegebenen Daten abhängen.

1. Der erste Ansatz erzeugt Entscheidungsbäume für die Trainingsaufgaben und berechnet anhand dieser Bäume bestimmte Kennzahlen, wie z.B. die Tiefe der Bäume. Die maximale Tiefe des Baumes gibt beispielsweise Aufschluss darüber, wie schwierig es ist, diese Aufgabe durch einen Entscheidungsbaum darzustellen. Das Verhältnis von Knotenanzahl zur Anzahl von Attributen hingegen zeigt die Anzahl irrelevanter Attribute an, es weist aber auch auf Attribute hin, die in mehr als einem Pfad des Baumes auftauchen. Weitere Kennzahlen sind: Knotenanzahl pro Datensatz, interne Symmetrie und Unausgeglichenheit. Diese Angaben stellen die Aufgabenbeschreibung dar und sind gleichzeitig die Eingabedaten des Meta-Lerners zusammen mit der Angabe, welcher der Lerner aus dem Lerner-Pool, die Trainingsaufgabe jeweils am besten gelöst hat.
   Der ENTRENCHER ist ein Meta-Lernsystem, das von Trainingsaufgaben generierte Entscheidungsbäume und die zugehörigen Informationen nutzt, um Klassifikationsaufgaben zu beschreiben (siehe [Ben98] und [Ben99]). Die Experimente, die in [Ben98] beschrieben werden, weisen erfolgreiche Ergebnisse bei der Auswahl von unterschiedlichen Lernoptionen mit Hilfe des ENTRENCHERs auf. Dabei ging es speziell um die Auswahl von Pruningstrategien.

2. Der zweite Ansatz (siehe [BGCK00]) basiert auf Meta-Lernen höherer Ordnung. Anstelle des alleinigen manuellen Berechnens bzw. Extrahierens der Informationen vom Entscheidungsbaum wie beim ersten Ansatz versucht dieser Ansatz zusätzlich, den Entscheidungsbaum direkt als Aufgabenbeschreibung zu nutzen. Für die Repräsentation des Baumes wird eine integrierte funktionale und logische Programmiersprache namens Escher verwandt. Die Auswahl der Informationen, die in einem Knoten dargestellt werden, ist aufgrund von bisherigen Erfahrungen mit Entscheidungsbäumen getroffen worden. Die Informationen eines inneren Knotens werden durch den Typ “InnererKnoten” zusammengefasst:

   ```
   type InnererKnoten = (Attribut, AnzahlBsp,
         Hauptklasse, AnzahlHk, Heuristik);
   ```

   Attribut gibt den Namen des in diesem Knoten getesteten Attributes an. *AnzahlBsp* ist die Anzahl der Datensätze, die diesen Knoten erreicht haben.

Hauptklasse ist diejenige Klasse, der die meisten Trainingdatensätze, die diesen Knoten erreicht haben, angehören. *AnzahlHk* gibt die Anzahl derjenigen Datensätze an, die diesen Knoten erreicht haben und der Hauptklasse angehören. *Heuristik* gibt z.B. den Informationsgewinn dieses Knotens an. Blattknoten stellen Sonderfälle des inneren Knotens dar, wobei Attribut und Heuristik nicht angegeben werden. Abbildung 4 zeigt den in [BGCK00] angegebenen Beispiel-Entscheidungsbaum. Zu erkennen ist unter anderem, dass die Trainingsdaten insgesamt 109 Instanzen enthalten und das nach zwei Klassen (True,False) klassifiziert wird. "NA" (not applicable) wird in den Blattknoten anstelle von Attribut und Heuristik angegeben.

Jede früher bearbeitete Aufgabe wird nun als Trainingsbeispiel durch einen

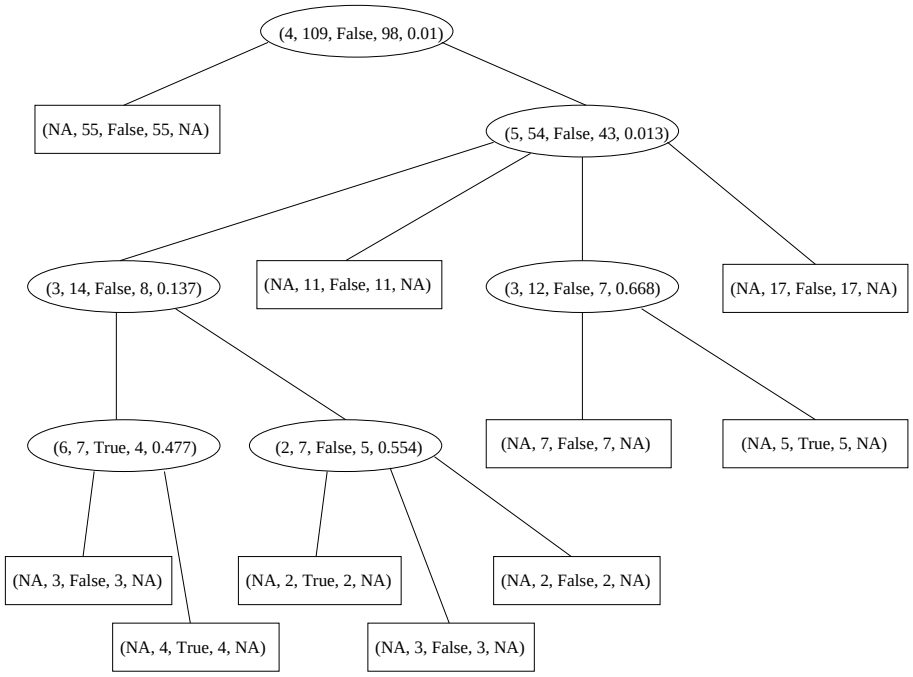


Abbildung 4: Beispiel-Entscheidungsbaum

Entscheidungsbaum, der auf diesen Typen "InnererKnoten" aufbaut, dargestellt und durch ein Label, dass den geeignetsten Algorithmus angibt, gekennzeichnet.

```
type MetaBeispiel = (Entscheidungsbaum, Label);
```

Nun wird mit Hilfe dieser Trainingsbeispiele und einem Lerner höherer Ordnung eine entsprechende Hypothese erstellt, die den bestgeeignetsten Algorithmus für eine neue Aufgabe bestimmt. [BGCK00] gibt eine Beispielre-

gel an, die den besten Algorithmus so auswählt: "Wenn Shape[2] einen Wert größer als 0.5 hat und es wenigstens zwei Kindsknoten des Wurzelknotens gibt mit Heuristik kleiner 0.003, dann nutze Backpropagation, andernfalls Naive Bayes."

Aus diesen Möglichkeiten der Aufgabenbeschreibung ergeben sich die unterschiedlichsten Metadaten, die zum Trainieren des Meta-Lerners eingesetzt werden können. Jedoch sind sie lediglich auf der Basis der Daten erstellt worden. Diese Art der Daten werden auch als datenbasierte Meta-Attribute bezeichnet. Wie bereits in [MST94] festgestellt wurde, hängen die Ergebnisse, die durch Anwenden einer Data-Mining-Technik erzielt werden, von weiteren Faktoren ab. Zu diesen Faktoren gehört unter anderem die eigentliche Implementierung eines Verfahrens, die zu unterschiedlichen Ergebnissen führen kann. Als Ergänzung zur Aufgabenbeschreibung schlägt [HK01] eine Beschreibung der Algorithmen vor, die z.B. Angaben über ihre Skalierbarkeit und ihre Toleranz gegenüber fehlenden Werten macht. Außerdem ist es das Ziel Nutzerpräferenzen zu berücksichtigen. Die nächsten beiden Abschnitte gehen auf diese beiden Punkte näher ein.

3.5 Algorithmenbeschreibung

In [HK01] wird die Beschreibung der Algorithmen als Ergänzung zu den datenbasierten Meta-Attributen betrachtet. Im wesentlichen versucht man dadurch, die "Area of Expertise" der einzelnen Algorithmen, siehe Abschnitt 3.4.2, konkreter zu beschreiben. Es ist eine Möglichkeit zur Vorauswahl von Algorithmen, die den Suchraum für ein geeignetes Modell einschränkt. Das heißt das Modellauswahl-Problem spaltet sich in zwei Teilprozesse auf. Im ersten Schritt (prior model selection) wählt man eine Teilmenge von in Frage kommenden Algorithmen mit Hilfe des Algorithmenprofils aus. In einem zweiten Schritt (posterior model selection) wird das Ergebnis mit Hilfe datenbasierter Meta-Attribute weiter eingeschränkt.
Die zentrale Frage dabei ist nicht, ob, sondern unter welchen Umständen ein Algorithmus gut bzw. schlecht arbeitet. Zur Erstellung eines Algorithmenprofiles werden in [HK00b] folgende Gesichtspunkte betrachtet:

Repräsentation und Funktionalität: Hier interessieren vor allem allgemeine Anforderungen, Fähigkeiten und Grenzen eines Lerners. Dazu gehören beispielsweise Angaben über die Attributtypen, die vom Algorithmus verarbeitet werden können, z.B. nominale, numerische oder beide. Wenn ein Algorithmus nur nominale Attribute als Eingabe akzeptiert, aber die Daten auch numerische Werte enthalten, dann scheidet dieser Algorithmus aus der Menge der geeigneten Algorithmen aus. Des weiteren kann es entscheidend sein, ob ein Algorithmus inkrementell

[2] Shape ist eine Kennzahl, die die Gestalt des Baumes beschreibt. Wie man diese Kennzahl berechnet, wird in Abschnitt 4.4.2 beschrieben.

lernen kann oder nicht und ob man über Parameter die Performanz des Lerners beeinflussen kann.

Effizienz: In diesem Punkt geht es speziell um Angaben zu Trainings- und Ausführungszeiten und um den Speicherplatz, der in diesen Phasen benötigt wird.
Die Trainingszeit gibt an, wie lange ein Lerner braucht, um ein bestimmtes Modell zu erzeugen. Dabei muss beachtet werden, ob die voreingestellten Werte für vorhandene Parameter genutzt werden oder ob extra Zeit investiert wird, um die optimalen Parametereinstellungen zu finden. Das Nutzen der Voreinstellungen reduziert die Trainingszeit erheblich, jedoch kann es zur Erhöhung der Fehlerrate führen. Offensichtlich ist jedenfalls, dass Verfahren, wie k-Nearest-Neighbor (k-NN), keine Trainingszeit beanspruchen, da sie die Trainingsinstanzen einfach nur abspeichern. Allerdings ist ihre Ausführungszeit erheblich schlechter als z.B. die von Entscheidungsbaumverfahren, diese brauchen bei der Ausführung nur wenige Eigenschaften zu testen. k-NN hingegen führt sämtliche Berechnungen über alle Trainingsinstanzen während der Ausführung durch. Die Ausführungszeit steigt in diesem Fall linear mit der Anzahl der Trainingsinstanzen, sofern keine Optimierungstechniken angewandt werden.
Den Speicherplatz betreffend stellt sich vor allem die Frage, ob es notwendig ist, dass die gesamten Daten im Hauptspeicher liegen oder ob der Algorithmus Partitionierungstechniken verwendet und auf Datenportionen arbeiten kann.

Robustheit (resilience): Die Robustheit eines Data-Mining-Algorithmus' sagt aus, wie empfindlich oder tolerant der Algorithmus auf unterschiedliche Eigenschaften der Daten reagiert. Im folgenden werden einige Eigenschaften von Algorithmen aufgezählt, die laut [HK00b] Aussagen zur Robustheit machen:

- Ein Lernalgorithmus ist skalierbar, wenn er sowohl für große, kleine und mittlere Datenmengen akzeptable Ergebnisse liefert. Die Skalierbarkeit kann zum einen auf die Anzahl der Instanzen und zum anderen auf die Anzahl der Attribute des Datensatzes bezogen werden. Im Fall von Klassifikationsaufgaben kommt die Anzahl der Klassen, die effizient gelernt werden können, noch hinzu.

- Die Fähigkeit mit fehlenden Werten umgehen zu können, ist in einigen Anwendungen besonders wichtig, z.B. wenn medizinische Diagnosen oder Umfragen ausgewertet werden. In einigen Fällen können fehlende Werte mit Hilfe von Hintergrundwissen ersetzt werden. Verbleibende fehlende Werte sollten den Lernalgorithmus jedoch nicht daran hindern akzeptable Modelle zu entwickeln. Wie bereits in Abschnitt 2.1 erwähnt wurde, sind Naive Bayes robust gegenüber fehlenden Werten, da sie für die Berechnung von Wahrscheinlichkeiten einfach ignoriert werden und keinen Einfluss auf das Ergebnis haben.

- Der Umgang mit verrauschten Daten, die z.B. durch Nutzen fehlerhafter Messgeräte entstehen, ist eine weitere Eigenschaft, die die Robustheit von Data-Mining-Algorithmen zum Ausdruck bringt. Besonders Verfahren, die mit Ähnlichkeitsmaßen arbeiten, sind anfällig gegenüber verrauschten Werten, sie können zu Falschklassifikationen führen. Entscheidungsbaumverfahren wirken dem mit Pruningtechniken entgegen (siehe [Ben98]). Hilario betrachtet auch das Vorhandensein irrelevanter Attribute als eine Form des Rauschens. Generell ist es möglich anzugeben, ob ein Algorithmus mit verrauschten Daten umgehen kann oder nicht. Allerdings ist es laut [HK00b] bisher nicht möglich, ein Maß für den Grad der Toleranz gegenüber verrauschten Daten anzugeben, um auf diese Weise die Algorithmen besser vergleichen zu können.

- Algorithmen können empfindlich auf redundante Attribute reagieren. Attribute werden als redundant bezeichnet, wenn sich z.B. die Werte eines Attributes durch die Werte eines oder mehrerer anderer Attribute bestimmen lassen, sie können auch als funktionale Abhängigkeiten bezeichnet werden. Es sei angemerkt, dass durch Experimente gezeigt wurde, dass Naive Bayes trotz der Unabhängigkeitsannahme der Attribute gute Ergebnisse erzielt (siehe [HK00b]).

Durchführbarkeit (practicality): Die Einschätzung, ob ein Algorithmus praktikabel ist oder nicht, hängt stark von den Vorlieben und Prioritäten des Nutzers ab. Deshalb ist die Einschätzung der Durchführbarkeit ein gutes Kriterium für die Vorauswahl von Algorithmen. Die Einschätzung wird dazu mit den Nutzerpräferenzen, auf die im nächsten Abschnitt eingegangen wird, verglichen.
Um z.B. die Handhabbarkeit eines Algorithmus' zu beurteilen, wird die Anzahl der vom Nutzer einzustellenden Parameter betrachtet. Für einen Laien erscheint sicherlich eine geringere Anzahl praktikabler, während andere Nutzer eine gewisse Anzahl von Parametern als Möglichkeit sehen, den Data-Mining-Prozess besser kontrollieren zu können.
Die Beurteilung der Interpretierbarkeit von Ergebnissen eines Verfahrens hängt vom Wahrnehmungsvermögen des Nutzers ab. In der Regel werden Entscheidungsbaumverfahren und Verfahren, die Regeln erzeugen, als leicht interpretierbar eingestuft, im Gegensatz zu Modellbäumen, die als weniger leicht zu interpretieren gelten. Des weiteren spielt es gerade beim Data Mining eine Rolle, ob die eigentliche Vorgehensweise des Verfahrens nachvollziehbar ist, um die Ergebnisfindung nachprüfen zu können und für andere zugänglich zu machen.

Als Quellen aus denen das Wissen über Algorithmen gewonnen werden kann, werden in [HK00b] folgende genannt: die Spezifikationen der Algorithmen, Expertenwissen und Erfahrungen aus früheren Untersuchungen.

3.6 Nutzerpräferenzen

Neben der Genauigkeit, etwas vorherzusagen bzw. zu beschreiben, und den Trainings- und Testzeiten können auch andere Kriterien zur Bewertung und Auswahl von Verfahren betrachtet werden, wie z.B. die Modellkomplexität und die Verständlichkeit. Diese Kriterien können mehr oder weniger wichtig für den Nutzer sein. Nutzerprofile helfen, den Bedürfnissen der Nutzer gerecht zu werden.
In [BPK] werden einige typische Nutzerprofile vorgestellt. Man unterscheidet darin folgende Nutzertypen:

- Experte auf dem Gebiet des Maschinellen Lernens und Data Mining
- Laie auf dem Gebiet des Maschinellen Lernens und Data Mining
- Wissenschaftler, Forscher oder Entwickler
- Geschäftsanwender, z.B. aus dem Marketing-, Produktions- oder Verkaufsbereich

Diesen Nutzertypen wurden dann Bewertungskriterien gewichtet zugeordnet, wobei nach sehr wichtig, wichtig, egal, weniger wichtig und irrelevant unterschieden wurde. In der Untersuchung wurden folgende vier Bewertungskriterien für Verfahren betrachtet: Genauigkeit, Zeitverbrauch, Modellkomplexität und Verständlichkeit. Die in einem Portfolio präsentierten Ergebnisse stammen von einer Umfrage, die von DaimlerChrysler durchgeführt wurde. Daraus wird ersichtlich, dass z.B. ein Geschäftsanwender, der Laie auf dem Data-Mining-Gebiet ist, viel Wert auf Genauigkeit und Verständlichkeit legt. Ein Wissenschaftler hingegen, der auch Laie auf dem Gebiet ist, legt großen Wert auf Genauigkeit, jedoch ist ihm der Zeitfaktor dabei weniger wichtig (siehe [BPK]).

Nachdem in den letzten Abschnitten verschiedene Ansätze zur Aufgabenbeschreibung, Gesichtspunkte zur Algorithmenbeschreibung und mögliche Nutzerprofile vorgestellt wurden, werden im nächsten Abschnitt konkrete Meta-Lerner beschrieben, die diese Möglichkeiten unterschiedlich nutzen.

3.7 Meta-Lerner zur Auswahl von Klassifikationsalgorithmen

In diesem Abschnitt werden drei Meta-Lerner vorgestellt, die im Rahmen von Forschungsarbeiten implementiert wurden. Sie wurden zunächst nur für die Auswahl von Klassifikationsalgorithmen entwickelt. Es werden ihre Hauptkomponenten genannt und es wird beschrieben, welche Art der Aufgabenbeschreibung sie verwenden und welches Verfahren zum Meta-Lernen eingesetzt wird. Jeder dieser Meta-Lerner betrachtet die gegebenen Klassifikationsalgorithmen nur mit ihren Default-Werten und nimmt keine Parametereinstellungen vor.

3.7.1 ENTRENCHER

In diesem Abschnitt wird die Arbeitsweise des bereits in Abschnitt 3.4.3 erwähnten ENTRENCHER von Bensusan vorgestellt. Er verwendet Entscheidungsbäume und damit verbundene Informationen zur Aufgabenbeschreibung. Der geeignetste Algorithmus wird durch einen Meta-Klassifizierer bestimmt.
Als Hauptkomponenten des ENTRENCHERs werden folgende genannt: ein Entscheidungsbaumverfahren sowie ein Beschreibungsvektor, der die Informationen, die aus dem Baum gewonnen werden enthält. Des weiteren gibt es eine Komponente, die den Beschreibungsvektor mit den Eigenschaften des Baumes füllt. Außerdem gibt es einen Bias-Pool (Lerner-Pool) und eine Komponente, die jeweils zu den Trainingsdaten den geeignetsten Algorithmus bestimmt. Die wichtigste Komponente ist der Meta-Klassifizierer.
Die Trainingsaufgaben werden zunächst mit einem Entscheidungsbaumverfahren ausgeführt, um den Beschreibungsvektor zu füllen. Als Entscheidungsbaumverfahren wird eine Variation des C4.5 eingesetzt. Es werden keine Pruningtechniken angewandt, um Informationsverlust zu vermeiden. Der gefüllte Beschreibungsvektor einer Aufgabe enthält dann Informationen, wie z.B. die Knotenanzahl pro Datensatz, die Knotenanzahl pro Attribut und die maximale Tiefe des Baumes.
Außerdem werden alle Klassifikationsverfahren des Lerner-Pools auf die Trainingsaufgaben angewandt, um den jeweils besten Algorithmus auswählen zu können. Eine Trainingsaufgabe wird mit demjenigen Klassifizierer gekennzeichnet, der die geringste Fehlerrate aufweist. Die Aufgabenbeschreibung wird also um ein entsprechendes Label ergänzt. Falls zwei Verfahren die gleiche Fehlerrate aufweisen, wird dasjenige Verfahren gewählt, welches das einfachere Modell erzeugt. Die Beschreibungsvektoren werden zusammen mit dem entsprechenden Label zum Trainieren und Testen des Meta-Klassifizierers genutzt, wobei die Label die zu lernenden Klassen darstellen. Das generierte Modell wird zum Klassifizieren von neuen Aufgaben genutzt. Als Meta-Klassifizierer verwendet der ENTRENCHER das C4.5 Verfahren mit Pruning.
Soll nun für eine neue Klassifikationsaufgabe ein geeigneter Algorithmus empfohlen werden, so wird zunächst ihr Beschreibungsvektor durch Ausführen des Entscheidungsbaumverfahren gefüllt. Die im Beschreibungsvektor enthaltenen Metadaten werden dann genutzt, um mit dem generierten Modell des Meta-Lerners die neue Aufgabe zu klassifizieren. Die der Aufgabe zugeordnete Klasse entspricht dem empfohlenem Algorithmus.

3.7.2 AST

Das von Lindner und Studer entwickelte AST (Algorithm Selection Tool) (siehe [LS99]) basiert im Gegensatz zum ENTRENCHER auf einem fallbasiertem Ansatz. Die Architektur von AST ist in Abbildung 5 dargestellt. Im wesentlichen führen drei Aspekte zur Entscheidung darüber, welcher Algorithmus ausgewählt

wird. Das sind zum einen die Einschränkungen durch die Algorithmen selbst, zum anderen die gegebenen Daten und schließlich die bisherigen Erfahrungen.
Die Einschränkungen, die durch die Algorithmen gegeben sind, werden durch ei-

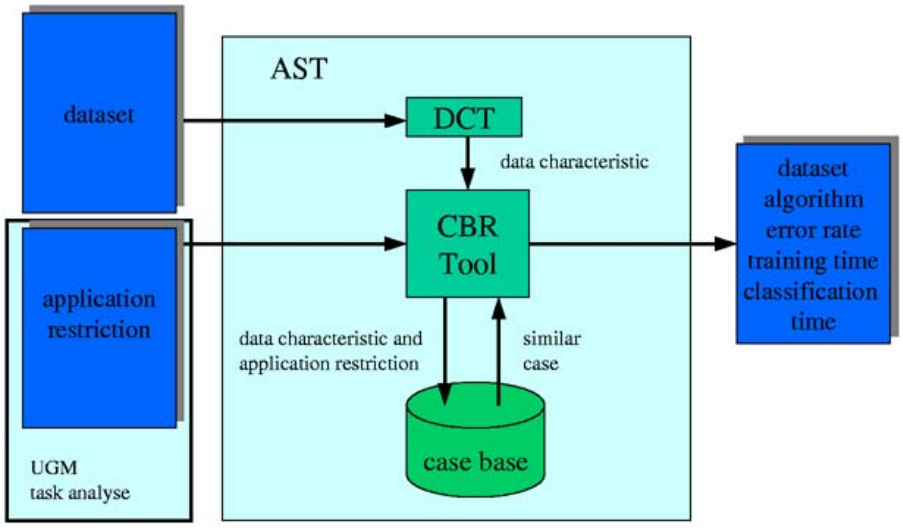


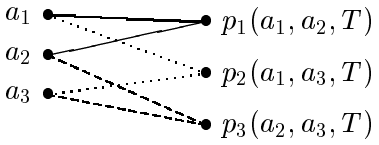
Abbildung 5: Architektur von AST

ne Aufgabenanalyse-Komponente, die Bestandteil eines Nutzerführungsmodules UGM (User Guidance Module) ist, analysiert. Im Grunde werden Profile von Algorithmen erstellt, die allgemeine Angaben enthalten über die Interpretierbarkeit der Modelle, über generelle Trainingszeiten (eingestuft nach schnell oder langsam) und über die Zugehörigkeit zu einer Algorithmenklasse. Folgende Algorithmenklassen werden unterschieden: Regellerner, Entscheidungsbäume, Neuronale Netze, Bayessche Netze und instanzbasierte Lerner. Des weiteren ist es dem Nutzer möglich durch diese Komponente, seine Präferenzen anzugeben.
Dateneigenschaften der gegebenen Daten werden durch eine Komponente namens DCT (Data Characterization Tool) berechnet. Ähnlich wie es im StatLog Projekt der Fall ist, werden allgemeine Kennzahlen, z.B. Attributanzahl, sowie informationstheoretische und statistische Informationen entsprechend dem Attributtyp (nominal oder numerisch) bestimmt.
Die bisherigen Erfahrungen umfassen Angaben, wie Fehlerrate und benötigte Trainings- und Testzeit, die bei der Anwendung eines bestimmten Algorithmus' auf einen gegebenen Datensatz ermittelt wurden. Bisher wurde allerdings nur die Fehlerrate für die weitere Ergebnissuche verwendet.
Entsprechend dieser drei Aspekte ist die Struktur der Fallbasis aufgebaut. Sie enthält demzufolge Attribute zur Beschreibung der Daten, der Algorithmen und den bisherigen Erfahrungen.
Wenn für eine neue Aufgabe ein geeigneter Algorithmus gefunden werden soll, dann gibt zunächst der Nutzer mit Hilfe des UGM seine Präferenzen bekannt und

das DCT bestimmt die Eigenschaften der gegebenen Daten. Beides zusammen stellt die Problembeschreibung dar. Danach berechnet AST die ähnlichsten Fälle durch Vergleichen der Dateneigenschaften mit denen, die in der Fallbasis enthalten sind. Die Lösungsbeschreibung umfasst also die in der Fallbasis gesammelten bisherigen Erfahrungen. Wenn der beste Algorithmus des ähnlichsten Falles auf die neuen Daten anwendbar ist, dann wird er als Empfehlung ausgegeben.

3.7.3 NOEMON

Das Analysesystem NOEMON [KT99] von Kalousis und Theoharis besteht aus zwei Hauptkomponenten. Die eine Komponente ist der "Assistant"(Assistent), sie stellt den Meta-Lerner dar. Die andere Komponente, der "Problem-Solver", führt den vom Assistenten empfohlenen Algorithmus für die gegebenen Daten aus, d.h. der "Problem-Solver" vollführt das eigentliche Data Mining. Im folgenden wird der Assistent näher beschreiben.
Der Assistent nutzt als Meta-Lern-Strategie das Nearest-Neighbour-Verfahren. Als Ergebnis liefert er ein Ranking der registrierten Klassifikationsalgorithmen. Dafür benötigt er folgende drei Komponenten: "NOEMON-META-LEARN", "NOEMON-RULEGENerator" und "NOEMON-SELector". Die Abbildung 6 aus [KT99] zeigt die Anordnung der drei Komponenten.
Zunächst werden durch "NOEMON-META-LEARN" die Meta-Lernprobleme erzeugt. Diese Meta-Lernprobleme haben eine etwas andere Struktur als die Trainingsbeispiele von AST und ENTRENCHER. Als erstes werden die Trainingsdatensätze mit den registrierten Klassifikationsalgorithmen des Lerner-Pools ausgeführt und die Werte der Vorhersagegenauigkeit bestimmt. Weitere Performanzkriterien, wie die Trainings- und Ausführungszeit sowie der Ressourcenbedarf, werden in der derzeitigen Version nicht genutzt, sie sind aber in [KT99] bereits angedacht. Des weiteren werden durch "NOEMON-META-LEARN" die Daten durch statistische und informationstheoretische Kennzahlen beschrieben, d.h. die Aufgabenbeschreibung wird auch durch diese Komponente generiert.
Nachdem die Metadaten und die Vorhersagegenauigkeitswerte ermittelt wurden, werden nun die eigentlichen Meta-Lernprobleme *p* festgelegt. Die Anzahl der Probleme ist abhängig von der Anzahl der registrierten Algorithmen. Da die Algorithmen paarweise betrachtet werden, erzeugt "NOEMON-META-LEARN" bei n Klassifikationsalgorithmen im Lerner-Pool $\binom{n}{2}$ Probleme. Bei drei gegebenen Algorithmen a_1, a_2 und a_3 ergibt sich dann folgendes:



Ein Problem p_i wird bestimmt durch ein Paar von Algorithmen und einer Menge von Daten *T*. Die Daten umfassen eine Menge von Klassifikationsaufgaben.

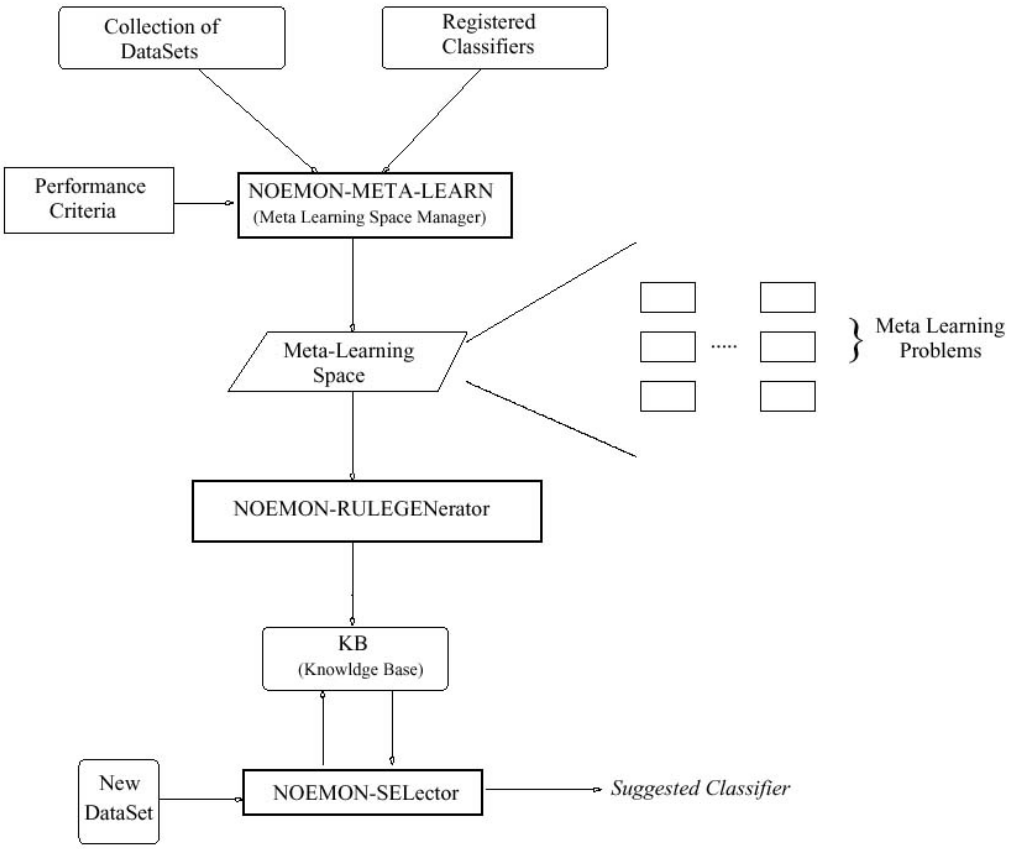


Abbildung 6: Komponenten des NOEMON

Sind a_x und a_y die betrachteten Algorithmen, und ist *T* die Menge der gegebenen Datensätze, dann kann man ein Problem darstellen als $p_i(a_x, a_y, T)$, wobei $1 \leq i \leq \binom{n}{2}$ und $1 \leq x \leq n$, $1 \leq y \leq n$ mit $x \neq y$. Jedes der Probleme p_i repräsentiert ein Meta-Lern-Problem.
Die Anzahl der Instanzen pro Problem ergibt sich aus der Anzahl der gegebenen Trainingsdaten. So ist beispielsweise $p_{i_j}(a_x, a_y, t_j)$ eine Instanz des Problems p_i angewandt auf den Datensatz t_j, wobei $t_j \in T$ und $1 \leq j \leq m$ gilt (*m* ist die Anzahl der Datensätze von *T*).
Nun werden die Performanzwerte der beiden Algorithmen a_x und a_y für t_j miteinander verglichen. Die der Instanz zugeordnete Klasse ist entweder der Algorithmus mit dem besseren Performanzergebnis oder "tie":

$$p_{i_j}(a_x, a_y, t_j) = \textit{Aufgabenbeschreibung von } t_j + (a_x |\ a_y |\ tie).$$

Letzteres deutet, wie auch beim Landmarking (Abschnitt 3.4.2), darauf hin, dass sich die einzelnen Performanzwerte nicht signifikant unterscheiden. Für jedes p_i werden alle Paare (Aufgabenbeschreibung(AB) + Klassenlabel) bestimmt. Betrachtet man wieder die Algorithmen a_1, a_2, a_3 und nimmt $T = \{t_1, t_2, t_3, t_4\}$ an, dann ergibt sich beispielsweise:

$$
\begin{aligned}
p_{1_1}(a_1, a_2, t_1) &= AB(t_1) + a_2 \\
p_{1_2}(a_1, a_2, t_2) &= AB(t_2) + a_2 \\
p_{1_3}(a_1, a_2, t_3) &= AB(t_3) + tie \\
p_{1_4}(a_1, a_2, t_4) &= \underbrace{AB(t_4) + a_1}
\end{aligned}
$$

wird zur Generierung der Regelmenge von p_1 genutzt

$$
\begin{aligned}
p_{2_1}(a_1, a_3, t_1) &= AB(t_1) + a_1 \\
p_{2_2}(a_1, a_3, t_2) &= AB(t_2) + a_3 \\
p_{2_3}(a_1, a_3, t_3) &= AB(t_3) + a_1 \\
p_{2_4}(a_1, a_3, t_4) &= \underbrace{AB(t_4) + a_3}
\end{aligned}
$$

wird zur Generierung der Regelmenge von p_2 genutzt

$$
\begin{aligned}
p_{3_1}(a_2, a_3, t_1) &= AB(t_1) + tie \\
p_{3_2}(a_2, a_3, t_2) &= AB(t_2) + a_3 \\
p_{3_3}(a_2, a_3, t_3) &= AB(t_3) + tie \\
p_{3_4}(a_2, a_3, t_4) &= \underbrace{AB(t_4) + a_2}
\end{aligned}
$$

wird zur Generierung der Regelmenge von p_3 genutzt

Auf jedes dieser Meta-Lern-Probleme p_i, dargestellt als eine Menge von Paaren (Aufgabenbeschreibung(AB) + Klassenlabel), wird nun das Nearest-Neighbour-Verfahren angewandt. Die entstehenden $\binom{n}{2}$ Regelmengen werden im Rahmen der "NOEMON-RULEGENerator"-Komponente generiert und in einer Wissensbasis abgelegt.
Wird jetzt dem NOEMON eine neue Klassifikationaufgabe gestellt, dann ermittelt die "NOEMON-SELector" Komponente zunächst die Metadaten, die diese Aufgabe beschreiben. Die Metadaten werden dann auf jede der in der Wissensbasis abgelegten Regelmengen angewandt. Aus der daraus resultierenden Auflistung von Algorithmen bzw. von "tie" wird ein Ranking erstellt. Der Algorithmus, der am häufigsten als Ergebnis aufgelistet ist und demzufolge am höchsten gerankt wird, stellt den empfohlenen Algorithmus dar.

3.8 Zusammenfassung

In diesem Kapitel wurde die Problematik des Meta-Lernens behandelt. Anfangs wurden verschiedene Ziele des Meta-Lernens im Data-Mining-Bereich vorgestellt

und dann wurde speziell auf das Ziel, das geeignetste Lernverfahren für eine Data-Mining-Aufgabe zu finden, eingegangen.
Es wurde ausgesagt, dass Meta-Lerner Verfahren des Maschinellen Lernens nutzen, um eine Zuordnung von Data-Mining-Aufgaben zu Lernverfahren zu finden. Besonders geeignet dafür sind Klassifikationsverfahren und das Fallbasierte Schließen.
Des weiteren wurde bemerkt, dass der Erfolg des Meta-Lernens zum einen vom verwendeten Meta-Lern-Verfahren abhängt und zum anderen von den Daten aus denen gelernt wird. Die Daten für den Meta-Lerner (Meta-Level-Attribute) ergeben sich aus den Metadaten, die die bisherigen Erfahrungen beschreiben. Die bisherigen Erfahrungen, aus denen gelernt werden soll, stellen bereits früher bearbeitete Aufgaben dar, für die das geeignetste Verfahren bekannt ist. Zur Ermittlung des geeignetsten Verfahren werden Evaluierungsstrategien angewandt, die vor allem die Fehlerrate zur Bewertung heranziehen.
Drei unterschiedliche Möglichkeiten zur Beschreibung von Klassifikationsaufgaben wurden vorgestellt:

- Der statistische und informationstheoretische Ansatz
- Das Landmarking
- Der Entscheidungsbaum-basierte Ansatz.

Bisher gibt es noch keinen Konsens, welche Strategie am besten zum Meta-Lernen geeignet ist. Des weiteren wurde gezeigt, dass auch Beschreibungen der einzelnen Verfahren in Form von Profilen sowie Nutzerpräferenzen die Zuordnung bzw. die richtige Auswahl des geeignetsten Algorithmus' beeinflussen.
Abschließend wurden drei Meta-Lerner vorgestellt, die im Rahmen von Forschungsarbeiten speziell für die Auswahl von Klassifikationverfahren entwickelt wurden. Jeder dieser Meta-Lerner repräsentiert eine andere Kombination von Meta-Lern-Verfahren (Meta-Klassifizierer, Fallbasiertes Schließen) und Art der Aufgabenbeschreibung (Entscheidungsbaum-basiert, statistisch und informationstheoretisch).

4 Konzeption eines Meta-Lerners

Ziel dieses Kapitels ist es in einem ersten Schritt, die allgemeinen Anforderungen an einen Meta-Lerner zu identifizieren. Mit “allgemeinen Anforderungen” ist in diesem Fall gemeint, dass sie sowohl für die Auswahl von Assoziations-, Klassifikations- und Clustering-Verfahren zutreffen. Im vorherigen Kapitel wurden bereits als Einführung in die Modellauswahlproblematik verschiedenste Ansätze des Meta-Lernens aufgezeigt. Es wurden konkrete Meta-Lerner zur Auswahl von Klassifikationsverfahren vorgestellt. Davon soll nun zunächst abstrahiert werden.
Im weiteren Verlauf dieses Kapitels wird darauf aufbauend eine Architektur entworfen, die die ermittelten Anforderungen adäquat erfüllt. Dazu werden notwendige Komponenten und ihr Zusammenwirken untereinander beschrieben. Diese Architektur wird dann speziell für einen Meta-Lerner zur Auswahl von Klassifikationsverfahren konkretisiert. Abschließend wird diskutiert, inwieweit sich diese konkrete Architektur unterscheidet von der eines Meta-Lerners zur Auswahl von Assoziations- bzw. Clustering-Verfahren.

4.1 Anforderungen an einen Meta-Lerner

Wie bereits in Kapitel 3 deutlich gemacht wurde, hat der Meta-Lerner die Aufgabe, den Nutzer bei der Auswahl von geeigneten Verfahren zur Lösung von Data-Mining-Aufgaben zu unterstützen. Der Nutzer erwartet, dass das vom Meta-Lerner empfohlene Verfahren die gestellte Aufgabe seinen Vorstellungen entsprechend bestmöglich löst. Aus dieser Erwartung heraus ergeben sich Anforderungen, die im folgenden dargelegt werden.

1. Es ist offensichtlich, dass der Meta-Lerner in der Lage sein muss, auf die Menge von Verfahren, die von dem Data Mining Tool zur Verfügung gestellt werden, zuzugreifen. Er muss sich einen Überblick über die im Lerner-Pool enthaltenen Algorithmen verschaffen und Informationen über ihre Fähigkeiten gewinnen, um den Suchraum der in Frage kommenden Verfahren einzuschränken.

2. Damit der Meta-Lerner in der Lage ist, eine Empfehlung für eine neue Aufgabe geben zu können, muss er bereits bearbeitete Aufgaben nutzen, und für diese Aufgaben muss bekannt sein, welches Verfahren zur Lösung am besten geeignet ist. Daher ist eine Evaluierungsstrategie notwendig.

3. Außerdem benötigt der Meta-Lerner eine Strategie, wie er die Aufgaben am besten beschreibt, d.h. die Art der Metadaten zur Aufgabenbeschreibung muss festgelegt werden. Weiterhin sollte die Beschreibung dieser Aufgaben und der neuen Aufgabe einheitlich erfolgen, damit ein Vergleich möglich ist.

4. Um für den Nutzer eine zufriedenstellende Auswahl treffen zu können, muss es möglich sein, Nutzerpräferenzen so anzugeben, dass sie bei der Verfahrensauswahl berücksichtigt werden.

5. Damit eine Modellauswahl stattfinden kann, benötigt der Meta-Lerner ein Verfahren, um eine Zuordnung von Aufgaben zu Algorithmen zu finden. Diese Zuordnung soll gewährleisten, dass für ähnliche Aufgaben, ähnliche Verfahren angewandt werden. Bisher ist die Auswahl eines solchen Verfahrens auf zwei generelle Alternativen beschränkt. Entweder der Meta-Lerner versucht, Zusammenhänge zwischen den bereits bearbeiteten Aufgaben, genauer gesagt zwischen ihren Beschreibungen und den zugehörigen besten Modellen aufzudecken oder er versucht, aus der Menge der bereits bearbeiteten Aufgaben eine ähnliche zur neuen Aufgabe zu finden, um von dem dort angegebenen Modell auf das jetzt gesuchte zu schließen. Im ersten Fall wird ein Klassifikationsverfahren benötigt, während im zweiten Fall die Methode des Fallbasierten Schließens anzuwenden ist.

6. Eine Evaluierungsstrategie ist notwendig, um den Erfolg des Meta-Lerners zu beurteilen.

7. Der Meta-Lerner soll effizient arbeiten, d.h. wenn es kostengünstiger ist, vorhandene Algorithmen durchzuprobieren, um den geeignetsten zu finden, dann hat der Meta-Lerner seine Existenzberechtigung verloren.

8. Des weiteren sollte der Meta-Lerner adaptive Fähigkeiten besitzen gegenüber Veränderungen im Lerner-Pool und dem Umfang der bisherigen Erfahrungen. Das Hinzufügen neuer Versionen von Algorithmen bzw. gänzlich neuer Verfahren und das Entfernen unnütz gewordener Methoden sollte von einem Meta-Lerner berücksichtigt werden. Derartige Veränderungen können die Überarbeitung sämtlicher bisheriger Erfahrungen erfordern. Es ist außerdem erstrebenswert, die Menge der bereits bearbeiteten Aufgaben inkrementell erweitern zu können, um dadurch eventuell zu neuen Erkenntnissen zu gelangen, d.h. die bisherigen Erfahrungen sollten schrittweise ergänzt werden können.

Aufgrund dieser Forderungen wurde eine Architektur erstellt. Ihre Komponenten und deren Aufgaben werden in den folgenden Abschnitten beschrieben.

4.2 Ein- und Ausgaben des Meta-Lerners

Aus der Beschreibung der Anforderungen haben sich bereits Kriterien für den Architekturaufbau eines Meta-Lerners ergeben, die in diesem Abschnitt aufgegriffen werden. So wurden bereits die grundlegenden Ein- und Ausgaben des Meta-Lerners genannt. Zu den Eingaben zählen:

- die Lernverfahren des Data Mining Tools, um eins davon auswählen zu können,
- die neue Aufgabe, um ein der Aufgabe entsprechend geeignetes Verfahren zu finden,
- die bereits bearbeitete Aufgaben, also Trainingsaufgaben, um aufgrund der bereits bekannten Zuordnung von Aufgaben zu Verfahren, ein geeignetes Verfahren für die neue Aufgabe zu empfehlen und
- die Nutzerpräferenzen, um die Auswahl entsprechend den Wünschen des Nutzers zu beeinflussen.

Die Ausgabe des Meta-Lerners ist ein Lernverfahren des Lerner-Pools, das für die neue Aufgabe und entsprechend der Nutzerpräferenzen als am geeignetsten ermittelt wurde. Abbildung 7 stellt die Ein- und Ausgaben des Meta-Lerners noch einmal dar.

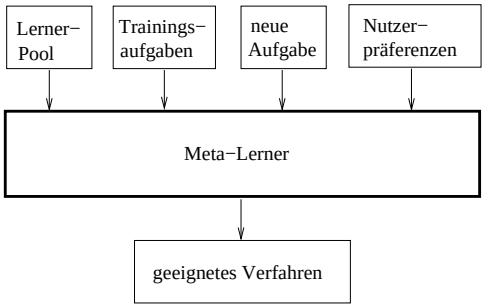


Abbildung 7: Ein- und Ausgaben des Meta-Lerners

Um mit Hilfe der Eingaben eine entsprechende Ausgabe generieren zu können, benötigt der Meta-Lerner Komponenten, die spezielle Aufgaben ausführen. Damit diese Komponenten identifiziert und ihre Aufgaben bestimmt werden können, wird zunächst der gesamte Auswahlprozess eines geeigneten Lernverfahrens dargestellt. Anhand der Abbildung 8 können die einzelnen Schritte während des Auswahlprozesses nachvollzogen werden.

4.3 Lernverfahrenauswahlprozess

Zunächst werden für die Data-Mining-Verfahren des Lerner-Pools die Lerner-Profile erstellt und mit Hilfe der bereits bearbeiteten Aufgaben die bisherigen Erfahrungen generiert. Auf den genauen Aufbau der Lerner-Profile und der bisherigen Erfahrungen wird später noch genauer eingegangen.
Nachdem die bisherigen Erfahrungen und die Lerner-Profile generiert wurden, kann eine neue Aufgabe gestellt werden. Dazu werden die dazugehörigen Daten

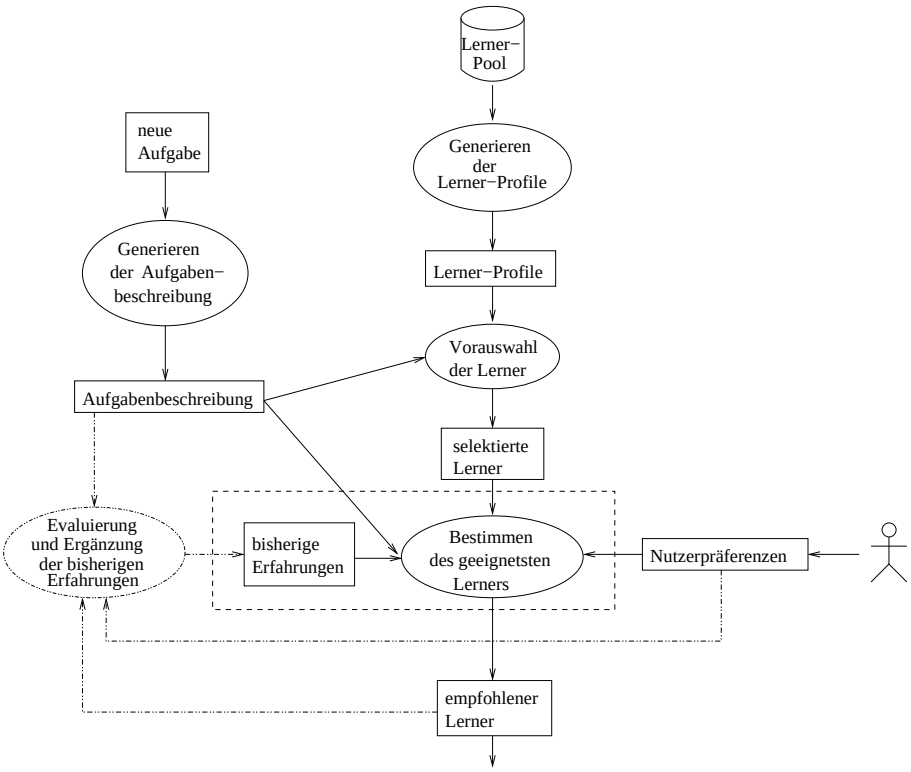


Abbildung 8: Auswahlprozess eines geeigneten Lernverfahrens

und Nutzerpräferenzen angegeben und es wird bekanntgegeben, um welchen Aufgabentypen es sich handelt (Klassifikation, Assoziation, Clustering). Dann wird anhand der Daten eine Charakterisierung der Aufgabe vorgenommen, d.h. die Aufgabenbeschreibung wird erzeugt.
Aufgrund der Art der Data-Mining-Aufgabe und der Beschaffenheit der Daten wird eine Vorauswahl der Lerner anhand ihrer Profile getroffen, wie es bereits in Abschnitt 3.5 beschrieben wurde (prior model selection). Dadurch wird der in Frage kommenden Suchraum für ein geeignetes Verfahren eingeschränkt. So ist z.B. sichergestellt, dass für das Clustering auch nur in der Menge der Cluster-Verfahren ein geeigneter Algorithmus gesucht wird. Ein weiterer Grund, der für die Vorauswahl von Lernverfahren spricht, ist, dass nicht jeder Algorithmus in der Lage ist, mit sämtlichen Datentypen zu arbeiten. Beispielsweise gibt es Klassifikationsverfahren, die nur eine Konzeptbeschreibung entwickeln können, wenn das Klassenattribut nominal (kategorisch) ist. Das bereits bekannte Wissen über die Verfahren wird auf diese Weise genutzt, um das Empfehlen von Algorithmen zu verhindern, die von vornherein unfähig sind, gewisse Aufgaben zu bearbeiten.
Die auf diese Weise selektierten Algorithmen gehen nun zusammen mit der Aufgabenbeschreibung, den Nutzerpräferenzen und den bisherigen Erfahrungen in die

eigentliche Verfahrensauswahl ein. Als Ergebnis wird das empfohlene Verfahren ausgegeben. Mit Hilfe einer Evaluierungsstrategie wird das Ergebnis bewertet. Gegebenenfalls kann es zusammen mit der Aufgabenbeschreibung und den Nutzerpräferenzen als Erweiterung der bisherigen Erfahrungen dienen. Die in Abbildung 8 durch durchgehende Linien gekennzeichneten Schritte, sind notwendig zur Bestimmung des geeignetsten Verfahrens, die durch Strich-Punkt-Linien gekennzeichneten Schritte dienen der Beurteilung und Verbesserung der Ergebnisse. Eine Verbesserung kann beispielsweise durch die Revidierung der bisherigen Erfahrungen erzielt werden.
In Abschnitt 3.2 wurde bereits darauf eingegangen, dass sowohl ein Meta-Klassifizierer als auch das Fallbasierte Schließen als Meta-Lernverfahren in Frage kommen. Wie die bisherigen Erfahrungen aufgebaut sind und auf welche Art das geeignetste Verfahren ermittelt wird, ist abhängig davon, welche dieser beiden Möglichkeiten realisiert wurde. Um die Vor- und Nachteile der Alternativen besser diskutieren zu können, werden sie im folgenden kurz vorgestellt. Die Abbildungen 9 und 10 repäsentieren dabei eine detailliertere Darstellung davon, was in Abbildung 8 durch ein gestricheltes Rechteck (bisherige Erfahrungen, Bestimmen des geeignetsten Lerners) gekennzeichnet ist.

4.3.1 Meta-Klassifizierer

Wenn ein Meta-Klassifizierer als Meta-Lernverfahren eingesetzt wird, werden die bereits bearbeiteten Aufgaben als Trainingsbeispiele verwendet, um ein Klassifikationsmodell zu erstellen. Dazu wird zunächst zu jeder dieser Aufgaben die Aufgabenbeschreibung generiert. Des weiteren ist an dieser Stelle bekannt, welches der im Lerner-Pool befindlichen Verfahren unter Beachtung von Nutzerpräferenzen am geeignetsten für diese Aufgaben ist. Ein Trainingsbeispiel setzt sich nun aus einer Aufgabenbeschreibung, den Nutzerpräferenzen und dem dazugehörigen geeigneten Verfahren zusammen. Letzteres stellt eine Ausprägung des zu lernenden Klassenattributes dar und wird in Abbildung 9, wie auch häufig in der Literatur, als Label bezeichnet. Die Aufgabenbeschreibung und die Nutzerpräferenzen stellen die Meta-Level-Attribute dar.

Um nicht nur den geeignetsten Algorithmus empfehlen zu können, sondern um auch Empfehlungen für Parameterbelegungen zu geben, sollte das Label um zutreffende Parameter und ihre Belegungen erweitert werden. Sinn macht dies allerdings nur, wenn die Parameterbelegungen sich von den Default-Werten der Algorithmen unterscheiden. Die maximale Anzahl möglicher Klassen entspricht demnach der Anzahl von Algorithmen mit ihren unterschiedlichen Parameterbelegungen.
Hat man nun diese Menge von Trainingsbeispielen gegeben, kann man den Meta-Klassifizierer darauf anwenden. Wie bereits erwähnt, könnte jedes der in Abschnitt 2.1 vorgestellten Verfahren eingesetzt werden. Wenn ein Entscheidungsbaumverfahren eingesetzt wird, ist das erzeugte Modell ein Entscheidungsbaum, im Falle

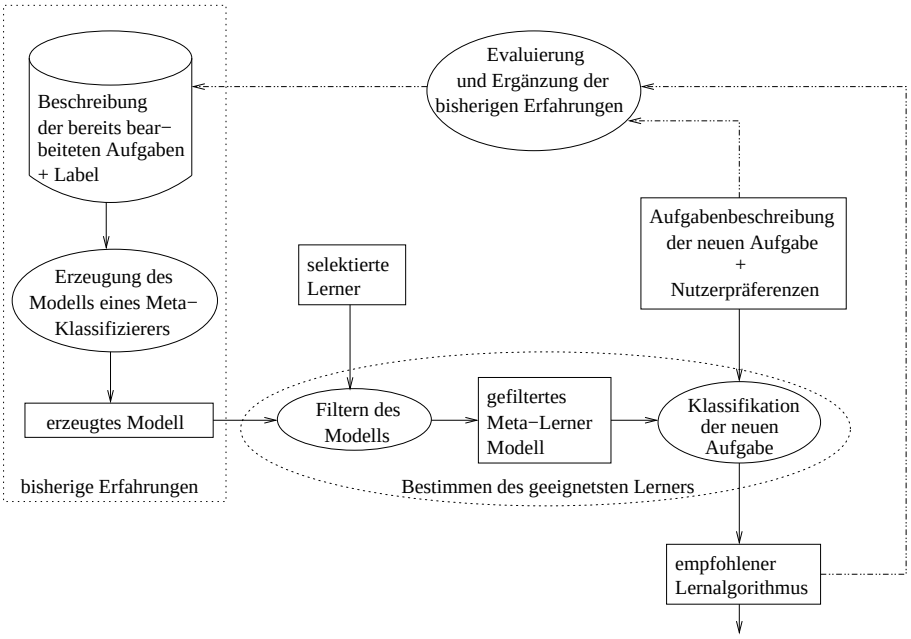


Abbildung 9: Klassifizierer als Meta-Lerner

eines Regellerners wäre es eine Menge von Regeln. Dieses erzeugte Modell ist die entscheidende Voraussetzung, um das Auswahlproblem zu lösen, denn es repräsentiert die gesuchte Zuordnung von Aufgaben zu Methoden, wie es in Kapitel 3 beschrieben wurde.
In einem nächsten Schritt wird dieses Modell anhand der bereits vorausgewählten Verfahren gefiltert. Eine Regelmenge wird dadurch auf diejenigen Regeln reduziert, die nur auf Verfahren verweisen, die in der Menge der vorausgewählten Verfahren enthalten sind. Im Fall eines Entscheidungsbaumes, werden die überflüssig gewordenen Zweige abgeschnitten. Auf dieses gefilterte Modell wird nun die neue Aufgabe angewandt, d.h. die Aufgabenbeschreibung und die Nutzerpräferenzen werden anhand dieses Modells klassifiziert. Die zugeordnete Klasse stellt den empfohlenen Lernalgorithmus mit seinen Parameterbelegungen dar.
Wie gut der Meta-Klassifizierer gelernt hat, kann, wie es bei der Evaluierung von Klassifizierern üblich ist, entweder durch Testdaten oder durch Kreuzvalidierung ermittelt werden (siehe Abschnitt 3.3). Um die gerade bearbeitete Aufgabe in die bisherigen Erfahrungen mit einfließen zu lassen, muss das Modell neu erzeugt werden. Bei Veränderungen im Lerner-Pool, müssen die Trainingsbeispiele neu getestet werden, d.h. es muss überprüft werden, ob die Zuordnungen von Aufgaben zu Verfahren immer noch zutreffen. Bei Änderungen muss auch an dieser Stelle das Modell neu erzeugt werden.

4.3.2 Fallbasiertes Schließen (CBR)

In [Ric95] identifiziert Richter vier Bestandteile, die in einem CBR-System für die Problemlösung von Bedeutung sind. Es sind:

- das Vokabular (Attribute, Prädikate ..),
- die Fallbasis,
- das Ähnlichkeitsmaß und
- die Lösungstransformation.

Diese Bestandteile finden sich in dem anhand von Abbildung 10 beschriebenen Prozess wieder. Das Vokabular legt fest, welche Attribute verwandt werden, z.B. um die Data-Mining-Aufgabe zu beschreiben. Die Fallbasis vereinigt die bisherigen Erfahrungen. Für jede bereits bearbeitete Aufgabe gibt es einen Eintrag in der Fallbasis. Ein Eintrag hat die gleiche Struktur, wie ein Trainingsbeispiel im Fall des Meta-Klassifizieres (siehe oben) und wird demzufolge auf gleiche Art generiert.
Diese Fallbasis wird, wie in Abbildung 10 erkennbar, gefiltert. Das bedeutet, die

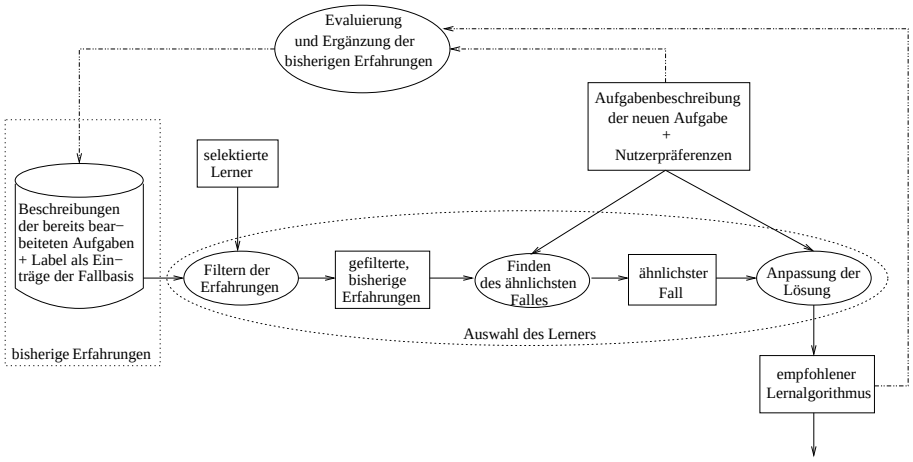


Abbildung 10: Meta-Lernen durch Fallbasiertes Schließen

Einträge werden reduziert, so dass nur noch diejenigen Einträge enthalten sind, deren Label in der Menge der bereits vorausgewählten Lernverfahren enthalten ist. Jetzt kann, ausgehend von dieser gefilterten Fallbasis, mit Hilfe des Ähnlichkeitsmaßes der ähnlichste Fall zur gestellten Data-Mining-Aufgabe bestimmt werden. Das Label des ähnlichsten Falles entspricht dem empfohlenen Verfahren. Mit Hilfe von Testdaten kann das Ergebnis des ähnlichsten Falles evaluiert werden.
Doch nun folgt noch der Schritt, den Richter als Lösungstransformation bezeichnet. An dieser Stelle können die Parametereinstellungen des Algorithmus' speziell

für die neue Aufgabe durch Adaptationsverfahren angepasst werden.
Um die gerade bearbeitete Aufgabe in die bisherigen Erfahrungen mit einfließen zu lassen, muss man lediglich die Fallbasis um einen Eintrag mit der Aufgabenbeschreibung, den Nutzerpräferenzen und dem entsprechenden Label erweitern. Im Gegensatz zur inkrementellen Erweiterbarkeit im Fall einer neuen Aufgabe musssen bei Veränderungen im Lerner-Pool, die Einträge der Fallbasis neu getestet werden, d.h. es muss überprüft werden, ob die Zuordnungen von Aufgaben zu Verfahren immer noch zutreffen.

4.3.3 Vergleich Meta-Klassifizierer und Fallbasiertes Schließen

Nachdem beide Möglichkeiten vorgestellt wurden, die als Meta-Lernverfahren in Frage kommen, werden sie nun miteinander verglichen.
Der fallbasierte Ansatz hat gegenüber dem Meta-Klassifizierer den Vorteil, dass er gerade für kleine Datenmengen gut geeignet ist. Aus Gründen des Datenschutzes und aufgrund von Betriebsgeheimnissen ist es schwierig, eine Vielzahl von Data-Mining-Aufgaben und speziell die dazu notwendigen Daten zu bekommen. Das umfassende Trainieren und Testen eines Meta-Klassifikationsmodells ist daher kaum möglich.
Der Vorteil eines Meta-Klassifikationsmodells liegt jedoch darin, dass es einmal generiert wird und zur Laufzeit des Meta-Lerners lediglich die Aufgabenbeschreibung der neuen Aufgabe und die Nutzerpräferenzen auf das Modell angewandt werden müssen. So wird recht schnell das empfohlene Verfahren als Ergebnis ausgegeben. Wohingegen beim Fallbasierten Schließen jedesmal die Einträge der Fallbasis zur Laufzeit mit Hilfe des Ähnlichkeitsmaßes durchsucht werden müssen, um den ähnlichsten Eintrag zur gegebenen Aufgabe zu finden. Eine effiziente Suche könnte jedoch durch ein entsprechendes Zugriffsverfahren auf die Fallbasis, beispielsweise durch ein mehrdimensionales Baumverfahren, ermöglicht werden (siehe [SH99]).
Die Möglichkeit der Parameteranpassung speziell für die gegebene Aufgabe, ist ein klarer Vorteil des Fallbasierten Schließens. Laut [ADL98] ist es mindestens genauso wichtig, Berechnungsaufwand in das Finden der besten Parameterbelegungen eines Algorithmus' zu investieren, wie in das Auswählen aus unterschiedlichen Algorithmentypen.
Das Erweitern der bisherigen Erfahrungen gestaltet sich beim fallbasierten Ansatz im Gegensatz zum Meta-Klassifizierer unkomplizierter, da es ein inkrementelles und kein one-shot Lernverfahren darstellt. Beim one-shot Lernverfahren wird das Konzeptmodell in einem Schritt über alle Trainingsdaten erzeugt, während ein inkrementelles Verfahren das Konzeptmodell schrittweise verfeinert. Der zu investierende Aufwand, bei Veränderungen im Lerner-Pool, ist dagegen bei beiden Ansätzen ähnlich.
Meta-Klassifizierer als auch Fallbasiertes Schließen wurden bereits in Forschungsprototypen, siehe Abschnitt 3.7, eingesetzt und in beiden Fällen war Meta-Lernen

möglich. In dem eben geführten Vergleich überwiegen jedoch die Vorteile des Fallbasierten Schließens und deshalb wird dieser Ansatz für den in dieser Arbeit zu konzipierenden Meta-Lerner eingesetzt.

Betrachtet man nun den gesamten Auswahlprozess von Abbildung 8 zusammen mit dem Fallbasierten Schließen, dann können folgende Komponenten des Meta-Lerners bestimmt werden:

- Komponente zum Generieren von Algorithmenprofilen (AP - Algorithm Profile),
- Komponente zum Aufbau der Fallbasis (CB - Case Base),
- Komponente zum Generieren von Aufgabenbeschreibungen (TD - Task Description),
- Komponente zum Bestimmen des ähnlichsten Falles (SC - Similar Case),
- Komponente zum Anpassen der Parameter des gefundenen Algorithmus' (AA - Adapt Algorithm).

Abbildung 11 zeigt die Komponenten noch einmal. Als Bezeichnung wurden die jeweils in Klammern angegebenen Abkürzungen verwendet.
Im nächsten Abschnitt werden die Aufgaben dieser Komponenten und ihr Zusam-

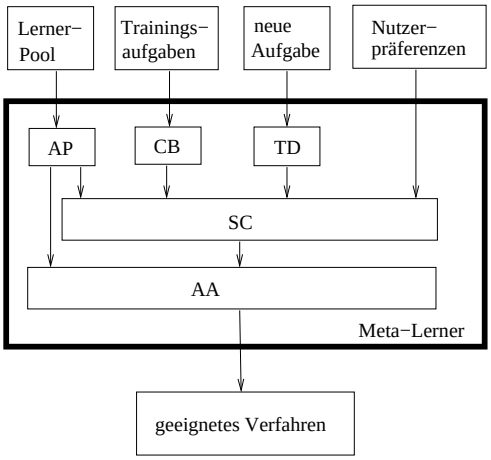


Abbildung 11: Komponenten des Meta-Lerners

menwirken untereinander konkretisiert und zwar speziell für das Auswählen von Klassifikationsverfahren.

4.4 Architektur eines Meta-Lerners zur Auswahl von Klassifikationsverfahren

In dieser Arbeit wird davon ausgegangen, dass ein Meta-Lerner sich dadurch auszeichnet, welche Meta-Level-Attribute und welches Verfahren zum Meta-Lernen genutzt werden. Wenn ein Meta-Lern-System sowohl Verfahren für die Klassifikation, die Assoziation und das Clustering empfehlen kann, so kann man sich demnach drei Meta-Lerner in einem zusammengefasst vorstellen. Abbildung 12

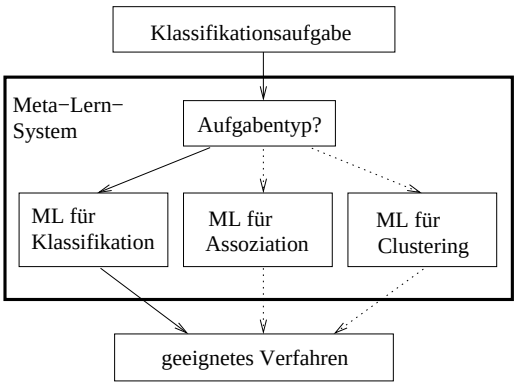


Abbildung 12: Meta-Lern-System

veranschaulicht diese Vorstellung. Man nimmt an, dass für jeden Aufgabentyp (Klassifikation, Assoziation und Clustering) eine spezielle Ausprägung eines Meta-Lerners angewandet werden muss, um bestmögliche Ergebnisse zu erzielen. Auch in [HK01] werden Aufgabentypen unterschieden, jedoch nur danach, ob sich überwachte oder unüberwachte Lernverfahren verlangen. Welche Unterschiede zwischen Meta-Lernern verschiedener Verfahrensfamilien auftreten können, wird in Abschnitt 4.5 diskutiert.
Für einen geplanten Meta-Lerner ist es wichtig, dass genügend potentiell einsetzbare Verfahren existieren, unter denen ausgewählt werden kann. Da in Data Mining Tools meistens Verfahren anzahlmäßig überwiegen, die den Klassifikationsverfahren zuzurechnen sind, wird in diesem Abschnitt ein Meta-Lerner zur Auswahl von Klassifikationsverfahren beschrieben. Dabei wird auf die zuvor bestimmten Komponenten, siehe Abbildung 11, genauer eingegangen.

4.4.1 Komponente zum Generieren von Algorithmenprofilen

Wenn ein Meta-Lerner an ein Data Mining Tool angebunden wird, müssen ihm Informationen über die darin enthaltenen Data-Mining-Verfahren zur Verfügung gestellt werden. Diese Informationen werden in einem Attributvektor, dem Algorithmusprofil, gespeichert. Die Registrierung eines Data-Mining-Verfahrens über

die AP-Komponente (Algorithm Profile) ist Voraussetzung dafür, dass ein Algorithmusprofil erstellt wird und dass das Verfahren dem Meta-Lerner als potentiell geeignetes Verfahren zur Auswahl steht.
Bereits in Abschnitt 3.5 wurden Eigenschaften genannt, die der Beschreibung von Algorithmen dienen. In diesem Fall interessieren vor allem Angaben über die Funktionalität der einzelnen Verfahren, denn die Profile werden zum einen zur Vorauswahl der Verfahren eingesetzt und zum anderen für die Lösungstransformation.
Klassifikationsverfahren unterscheiden sich in ihrer Funktionalität besonders dahingehend, welche Skalentypen für Attribute zulässig sind. Dieses Kriterium ist besonders für die Vorauswahl der Verfahren interessant. [Noc00] erläutert eine Reihe von Skalentypen, die meisten Data-Mining-Verfahren unterscheiden jedoch hauptsächlich zwischen numerischen und nominalen Attributen. Dem Skalentyp des Klassenattributes, welches gelernt werden soll, gilt dabei besonderes Interesse.
Für den Schritt der Lösungstransformation, ist es wichtig, dass das Lerner-Profil Angaben über mögliche Parameter enthält. Wenn ein Algorithmus Parameter besitzt, die die Performanz beeinflussen, so werden diese dem Lerner-Profil hinzugefügt. Im Fall von numerischen Parametern werden Minimum- und Maximum-Werte festgehalten und bei nominalen Parametern werden die einzelnen Ausprägungen aufgelistet. Flags, also Parameter, die entweder gesetzt sein können oder nicht, werden aufgezählt.
Im Interesse des Nutzers sollte das Algorithmenprofil auch einen Eintrag über die Interpretierbarkeit des vom Algorithmus erzeugten Klassifikationsmodelles enthalten. Dabei soll zwischen leicht und weniger leicht interpretierbar unterschieden werden. Entscheidungsbäume und Regelmengen werden beispielsweise als leicht interpretierbar eingestuft, M5-Bäume (siehe Abschnitt 2.1) hingegen als weniger leicht. Im Abschnitt 4.4.3 zur CB-Komponente wird beschrieben, wie die Interpretierbarkeit in die Auswahl des geeignetsten Algorithmus eingeht.
Abschließend sei noch einmal zusammengefasst, welche Informationen ein Lerner-Profil enthält:

- mögliche Skalentypen der Attribute, die zur Erstellung des Klassifikationsmodelles führen,
- mögliche Skalentypen des Klassenattributes (Attribut, nachdem klassifiziert wird),
- Angaben zu Parametern, die die Performanz beeinflussen,
- Einschätzung der Interpretierbarkeit.

Betrachtet man beispielsweise den in Abschnitt 2.1 vorgestellten ID3-Klassifikationsalgorithmus von Quinlan, so ist dieser nur in der Lage mit nominalen Attributen zu arbeiten, das gilt auch für das Klassenattribut. Parameter, die die Performanz beeinflussen, sind nicht gegeben. Die Weiterentwicklung hingegen, der

C4.5-Algorithmus, kann bis auf das Klassenattribut auch mit numerischen Attributen arbeiten. Des weiteren kann mittels Parameter angegeben werden, ob der zu erzeugende Entscheidungsbaum beschnitten werden soll oder nicht. Wenn der Baum beschnitten werden soll, kann der Nutzer die Vertrauensschwelle für das Pruning angeben. Das von beiden Algorithmen erzeugte Klassifikationsmodell wird als leicht interpretierbar eingestuft, da es ein Entscheidungsbaum bzw. eine daraus erzeugt Regelmenge ist.
Nachdem dargestellt wurde, wie die Algorithmen aus dem Lerner-Pool mit Hilfe der AP-Komponente durch Lerner-Profile beschrieben werden, geht es nun um die Aufgabenbeschreibungen.

4.4.2 Komponente zum Generieren von Aufgabenbeschreibungen

Um eine Zuordnung von Klassifikationsaufgaben zu Klassifikationsverfahren finden zu können, werden die Aufgaben mit Hilfe von daten-basierten Charakteristika beschrieben. Diese Charakteristika sind Daten über die Daten der Aufgaben, sie können deshalb auch als Metadaten bezeichnet werden. Die TD-Komponente (Task Description) ist für das Generieren dieser Metadaten verantwortlich.
In Abschnitt 3.4 wurden bereits die drei existierenden Ansätze vorgestellt, wie Klassifikationsaufgaben beschrieben werden können. Das Landmarking ist ein interessanter Ansatz, erfordert aber hohen Aufwand, die Landmarker selbst und ihre Anzahl zu bestimmen, siehe Abschnitt 3.4.2.
In der Regel wird der statistische und informationstheoretische Ansatz angewandt. Abgesehen von dem Berechnungsaufwand einiger Statistiken gibt es jedoch das Problem, dass manche Kennzahlen nur für nominale bzw. nur für numerische Attribute berechnet werden können. Wenn die Daten einer Klassifikationsaufgabe nun beispielsweise keine numerischen Attribute enthält, so werden die Charakteristika, die ausschließlich der Beschreibung numerischer Attribute dienen, wie z.B. die Verteilung der Attribute, als “nicht anwendbar” (not applicable) gekennzeichnet. Das Problem liegt nun darin zu bestimmen, wie ähnlich sich zwei Datensätze sind, die Attribute unterschiedlicher Skalentypen besitzen. In [SB00] wird in diesem Fall, die maximale Distanz (1) zugewiesen.
Aus diesem Grund wird die Entscheidungsbaum-basierte Aufgabenbeschreibung ausgewählt. Dieser Ansatz liefert einen homogenen Beschreibungsvektor bestehend aus zehn numerischen Attributen, die für alle Daten von Klassifikationsaufgaben berechnet werden können. Außerdem ist dies ein Ansatz speziell für Klassifikationsaufgaben, denn es wird ein Entscheidungsbaumverfahren zum Generieren der Metadaten genutzt. Die Gestalt des aus den Daten erzeugten Entscheidungsbaumes (ohne Anwendung von Pruningtechniken) stellt implizit die Charakteristika der Aufgabe dar. In Abschnitt 3.4.3 wurden bereits einige der Meta-Attribute genannt, die anhand des Baumes ermittelt werden. An dieser Stelle werden die hier genutzten Kennzahlen und ihre Ermittlung, wie sie von Bensusan in [Ben99] vorgeschlagen werden, noch einmal erläutert:

- **Knoten pro Attribut** (nodes per attribute) ist das Verhältnis aus Knotenanzahl des Baumes und Attributanzahl. Es zeigt an, wieviele Attribute in mehr als einem Zweig des Baumes für die Trennung der Daten genutzt werden.

- **Knoten pro Instanz** (nodes per instance) ist das Verhältnis aus Knotenanzahl des Baumes und Anzahl der Trainingsinstanzen. Es zeigt die Anzahl der Verzweigungen, die die Trainingsdaten erfordern.

- **Durchschnittliche Blattstärke** (average leaf corroboration) ist die durchschnittliche Stärke, mit der jedes Blatt des Baumes unterstützt wird. Wie stark ein Blatt unterstützt wird, hängt von der Anzahl der Trainingsinstanzen ab, die nach Durchlaufen des Baumes in dem Blatt enden. Diese Kennzahl beabsichtigt zu messen, wieviel Unterstützung jedes Element des Baumes von den Daten erhält.

- **Durchschnittliche Gewinnquoten-Differenz** (average gain-ratio difference) ist die durchschnittliche Differenz der Gewinnquoten zwischen den Attributen und zwar vor der ersten Verzweigung während des Aufbauprozesses des Entscheidungsbaumes. Die Gewinnquote ist ein informationstheoretisches Maß, das bei Entscheidungsbaumverfahren genutzt wird, um ein Attribut als Trennattribut für eine Verzweigung des Baumes auszuwählen. Angenommen, die Daten enthalten *n* Attribute, dann wird die durchschnittliche Gewinnquoten-Differenz wie folgt berechnet:

$$\left| \frac{1}{n-1} \sum_{i=2}^{n} \textit{gain-ratio}(a_i, T) \; - \; \textit{gain-ratio}(a_{i-1}, T) \right|$$

  wobei von einer absteigenden Sortierung der Gewinnquoten ausgegangen wird. Mit *a* sind die Attribute bezeichnet, und *T* repräsentiert die Aufgabe (Task), der die Attribute zuzuordnen sind. Wie die einzelnen Gain-Ratio-Werte berechnet werden, kann im Anhang B nachgelesen werden.
  Ist die Differenz groß, weist das darauf hin, dass es ein Attribut gibt, von dem das Klassenattribut sehr stark abhängt.

- **Maximale Tiefe** (Maximum depth) gibt die Größe des längsten Pfades von der Wurzel des Baumes zu einem Blatt an. Die Größe eines Pfades wird an der Anzahl von Knoten, die er enthält, gemessen. Die Tiefe zeigt an, wie schwer es ist, die Aufgabe mit Hilfe eines Entscheidungsbaumes darzustellen.

- **Anzahl sich wiederholender Knoten** (number of repeated nodes) gibt an, wieviele sich wiederholender Attribute im Baum auftreten. Dadurch wird angezeigt, wie zugänglich die Daten gegenüber linearen Aufteilungen sind, denn sie sind ein guter Indikator dafür, ob die Daten sehr fragmentierte Regionen enthalten, die Zerlegungen schwierig machen.

- **Gestalt** (shape) ist eine Funktion der Wahrscheinlichkeiten, an einem beliebigen Blatt anzukommen, wenn ein zufälliger Weg durch den Baum gewählt wird. $p(N_i)$ sei die Wahrscheinlichkeit zu einem Knoten N_i von dem Vorgänger N_A aus zu gelangen, wobei N_i einer von *m* möglichen Kindsknoten ist.

  $$p(N_i) = p(N_A)/m$$

  Wenn $p(N_A) = 1$, dann ist *A* die Wurzel des Baumes.
  Die Gestalt eines Baumes mit *n* Blättern wird dann wie folgt berechnet, wobei $p(L_j)$ die Wahrscheinlichkeit der Blätter darstellt:

  $$\textit{shape} = -\sum_{j=1}^{n} p(L_j) log_2(p(L_j))$$

  Diese Kennzahl gibt an, wie die Zerlegungen des Baumes aufeinanderfolgen.

- **Homogenität** (homogeneity) ist die Anzahl der Blätter geteilt durch die Kennzahl "Gestalt". Sie zeigt, wie die Blätter innerhalb des Baumes verteilt sind.

- **Unausgeglichenheit** (imbalance) wird wie folgt berechnet:
  Wenn alle möglichen Werte V_i für p(L_j) gegeben sind, kann G(V_i) als

  $$G(V_i) = nV_i$$

  definiert werden, wobei *n* die Häufigkeit ist, mit der V_i in der Menge aller Blätter des Baumes auftritt. Die Unausgeglichenheit berechnet sich dann aus der Summe über alle möglichen Werte für p(L_j).

  $$\text{imbalance} = -\sum_{j=1}^{n} G(V_j) log_2(G(V_j))$$

  Die Unausgeglichenheit zeigt an, wie gut ein einzelner Lernalgorithmus dafür geeignet ist, diese Aufgabe zu lösen.

- **Innere Symmetrie** (internal symmetry) ist die Anzahl der sich wiederholenden Teilbäume. Sie offenbart die Notwendigkeit von Attributkombinationen, denn sie zeigt, wie oft Attributgruppen zusammen im Baum auftreten.

Zur Verdeutlichung werden die Meta-Attribute für einen Datensatz beispielhaft durchgerechnet. Als Beispieldatensatz dient der Wetter-Datensatz, der im Anhang A zu finden ist.

Attribut	Ausprägung
outlook	sunny, overcast, rainy
temperature	numeric
humidity	numeric
windy	true, false
play	yes, no

Tabelle 1: Attribute des Wetter-Datensatzes

Beispiel: Der Wetter-Datensatz besteht aus 4 Attributen (outlook, temperature, humidity, windy) und dem Klassenattribut (play) und 14 Instanzen. Die Ausprägung der Attribute gestaltet sich, wie in Tabelle 1 dargestellt. Die Klassifikationsaufgabe besteht darin herauszufinden, ob ein Spiel gespielt werden kann oder nicht. Das Spiel wird draußen gespielt, deshalb wird die Entscheidung von den in Tabelle 1 angegebenen Attributen, die das Wetter beschreiben, abhängig gemacht.
Der mit Hilfe des C4.5-Algorithmus generierte Entscheidungsbaum ist in Ab-

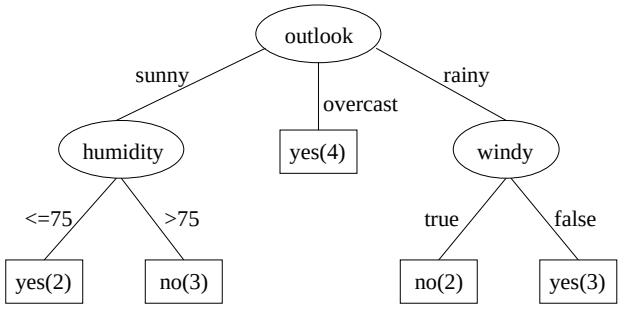


Abbildung 13: Entscheidungsbaum des Wetter-Datensatzes

bildung 13 dargestellt. Anhand dieses Baumes werden nun die Werte der Meta-Attribute berechnet.

- **Knoten pro Attribut:**

$$\frac{\textit{Knotenanzahl}}{\textit{Attributanzahl}} = \frac{8}{4} = 2$$

- **Knoten pro Instanz:**

$$\frac{\textit{Knotenanzahl}}{\textit{Instanzanzahl}} = \frac{8}{14} \approx 0.57$$

- **Durchschnittliche Blattstärke:**

$$\frac{\textit{Blattstärken}}{\textit{Blattanzahl}} = \frac{2+3+4+2+3}{5} = 2.8$$

- **Durchschnittliche Gewinnquoten-Differenz:** Die Gain-Ratio-Werte der Attribute wurden nach den im Anhang B angegebenen Formeln berechnet und in Tabelle 2 angegeben. Die Werte in der Tabelle wurden bereits absteigend

Attribut	Gain-Ratio-Wert
outlook	0.1564
windy	0.0488
temperature	0
humidity	0

Tabelle 2: Gain-Ratio-Werte der Attribute des Wetter-Datensatzes

sortiert.

$$\left| \frac{1}{3} \left(0.0488 + 0 + 0\right) - 0.1564 \right| \approx 0.1401$$

- **Maximale Tiefe:** 3
- **Anzahl sich wiederholender Knoten:** 0
- **Gestalt:** Die Tabelle 3 listet die Wahrscheinlichkeiten der Knoten auf, die Blätter sind fett hervorgehoben.

$$shape = -\left(\frac{1}{6} log_2 \left(\frac{1}{6}\right) + \frac{1}{6} log_2 \left(\frac{1}{6}\right) + \frac{1}{3} log_2 \left(\frac{1}{3}\right) + \frac{1}{6} log_2 \left(\frac{1}{6}\right) + \frac{1}{6} log_2 \left(\frac{1}{6}\right) \right) \approx 2.2516$$

Knoten *N*	*p(N)*
outlook	1
humidity	1/3
yes(2)	1/6
no(3)	1/6
yes(4)	1/3
windy	1/3
no(2)	1/6
yes(3)	1/6

Tabelle 3: Wahrscheinlichkeiten der Knoten

- **Homogenität:**

$$\frac{\textit{Blattanzahl}}{\textit{Gestalt}} = \frac{5}{2.2516} \approx 2.2206$$

- **Unausgeglichenheit:** Für die Berechnung der Gestalt wurden bereits die Wahrscheinlichkeiten der Blätter ermittelt. Zwei mögliche Werte *V* der Wahrscheinlichkeiten können angegeben werden: 1/6 und 1/3, wobei 1/6 viermal und 1/3 einmal auftritt. Die sich daraus ergebenden Werte für *G(V)* sind in Tabelle 4 angegeben.

p(L)	*G(V)*
1/6	4/6
1/3	1/3

Tabelle 4: Ermittelte G(V)-Werte

$$imbalance = -\left(\frac{4}{6}\, log_2\left(\frac{4}{6}\right) + \frac{1}{3}\, log_2\left(\frac{1}{3}\right)\right) \approx 0.9183$$

- **Innere Symmetrie:** 0

Neben dem Berechnen dieser Kennzahlen hat die TD-Komponente noch eine zweite Aufgabe. Sie bestimmt zusätzlich die Skalentypen der Attribute der Daten. Diese Angaben werden bei der Vorauswahl der Algorithmen mit den Lerner-Profilen verglichen, um somit funktional ungeeignete Algorithmen auszuschließen. Fasst man die Aufgabe der TD-Komponente noch einmal zusammen, so führt sie ein Entscheidungsbaum-Verfahren auf den Daten der Klassifikationsaufgabe aus, berechnet die eben aufgelisteten Metadaten und gibt diese aus. Zusätzlich bestimmt sie die Skalentypen der Attribute.

4.4.3 Komponente zum Aufbau der Fallbasis

Da das Fallbasierte Schließen als Methode zum Meta-Lernen gewählt wurde, ist es notwendig, eine Fallbasis aufzubauen. Die Fallbasis enthält sozusagen das Wissen der bisherigen Erfahrungen. Die CB-Komponente (Case Base) ist für das Generieren der Fallbasis-Einträge verantwortlich.
Die Fallbasis enthält einen Eintrag pro Aufgabe. Die Grundstruktur eines Eintrages setzt sich zusammen aus:

- den Metadaten der Aufgabenbeschreibung,
- den Nutzerpräferenzen
- sowie dem geeignetsten Verfahren für die Aufgabe.

Die Metadaten und die Nutzerpräferenzen beschreiben einen Fall, während das geeignetste Verfahren die Lösung des Falles darstellt. Die Metadaten der Aufgabenbeschreibung werden von der TD-Komponente angefordert. Die Nutzerpräferenzen werden von der Eingabe des Meta-Lerners übernommen. Der Nutzer kann

zum einen angeben, ob der geeignete Klassifikationsalgorithmus ein leicht zu interpretierendes Modell erzeugen soll. Zum anderen kann er darauf Einfluss nehmen, wie stark die Zeit gegenüber der Erfolgsrate bei der Bestimmung des geeignetsten Verfahrens gewichtet werden soll. Darauf wird im folgenden noch genauer eingegangen.
Ist das geeigneste Verfahren für die Aufgabe, die in die Fallbasis aufgenommen werden soll, bereits bekannt, so ist der Eintrag damit vollständig und kann hinzugefügt werden. Oftmals ist es jedoch so, dass die bisherigen Erfahrungen erst gewonnen werden müssen. Dazu werden der CB-Komponente eine Menge von Klassifikationsaufgaben zur Verfügung gestellt.
Für jede dieser Aufgaben wird nun eine Vorauswahl der durch die AP-Komponente registrierten Algorithmen durchgeführt. Die durch die TD-Komponente bestimmten Skalentypen der Attribute werden genutzt, um sie mit den in den Lerner-Profilen angegebenen möglichen Skalentypen zu vergleichen. Sollte ein Algorithmus nicht in der Lage sein, die Attribute der Aufgabe zu verarbeiten, so wird dieser im weiteren Auswahlprozess nicht betrachtet. Alle übrigen, vorausgewählten Algorithmen werden auf die Aufgabe angewandt und es wird ihre Performanz bestimmt.
In Abschnitt 3.3 wurde bereits darauf eingegangen, dass die Fehlerrate allein nicht aussagekräftig genug ist, um die Performanz von Algorithmen zu bewerten. Die Zeit, die das Lernverfahren zur Modellerstellung (Trainingszeit + Testzeit) benötigt, soll deshalb zusätzlich mit einfließen. Es wurden außerdem schon zwei Evaluierungsstrategien genannt, die sowohl die Fehlerrate als auch die Zeit in ihrer Bewertung einbeziehen. Die beiden werden im folgenden genauer beschrieben und verglichen, um anschließend eine davon auszuwählen.

A Simple and Intuitive Measure (SIM): Die Grundidee der in [SCB] vorgestellten SIM-Strategie ist eine recht einfache. Im Prinzip vergleicht sie die Performanzwerte der vorausgewählten Verfahren aus dem Lerner-Pool mit denen eines idealen Algorithmus'. Der ideale Algorithmus *o* erfüllt immer die optimale Erfolgsrate von 100% und benötigt dafür die optimale Zeit von 0 Zeiteinheiten. Die Differenzen der Erfolgsraten (SR) und Zeiten (T) werden miteinander multipliziert, um den SIM-Wert zu ermitteln. Der SIM-Wert für einen Algorithmus *a* wird wie folgt berechnet:

$$SIM_a = |(SR_a - SR_o)(T_a - T_o)| \,.$$

Die allgemeine Formel kann aber auch andere Kriterien einbeziehen, sofern man ein Optimum dafür angeben kann. Die allgemeine Formel lautet:

$$SIM_a = \prod_c |v_{c,a} - o_c| \,,$$

wobei *c* ein Kriterium (Erfolgsrate,Zeit) und $v_{c,a}$ der Performanzwert von *a* entsprechend *c* ist. o_c ist der optimale Wert für *c*. Derjenige Algorithmus, dessen SIM-Wert am kleinsten ist, stellt den geeignetsten Algorithmus dar.
Damit Nutzerpräferenzen die Auswahl von Verfahren beeinflussen können, wurde

die SIM-Strategie erweitert zur Bounded-SIM-Strategie, kurz BSIM. Dem Nutzer wird es ermöglicht für jedes Evaluierungskriterium einen Grenzwert vorzugeben. Überschreitet ein Algorithmus einen einzigen der vorgegebenen Grenzwerte, dann wird ihm ein unendlicher Wert zugeordnet, und er fällt damit aus der Menge der geeigneten Algorithmen heraus:

$$BSIM_a = \begin{cases} SIM_a & if\ \forall c\ \ v_{c,a} \in [b_c, o_c], \\ \infty & otherwise. \end{cases}$$

Gibt es mehrere Algorithmen, deren Performanzwerte innerhalb der Grenzen liegen, dann ist der mit dem kleinsten SIM-Wert der geeignetste Algorithmus.

Adjusted Ratio of Ratios (ARR): Die in [SB00] vorgestellte ARR-Methode erstellt ein Ranking der Algorithmen. Das Ranking wird direkt von den wie folgt berechneten ARR-Werten der Algorithmen abgeleitet. Der Rankwert eines Algorithmus' a_p ergibt sich aus der Summe aller paarweise ermittelten ARR-Werte dividiert durch die Anzahl der vorausgewählten Algorithmen vermindert um eins:

$$ARR_{a_p} = \left(\sum\nolimits_{a_q} ARR_{a_p,a_q} \right) / (m-1).$$

Die Performanz eines Algorithmus' a_p wird mit der Performanz aller anderen vorausgewählten Algorithmen a_q ins Verhältnis gesetzt. ARR_{a_p,a_q} ergibt sich aus:

$$ARR_{a_p,a_q} = \frac{\frac{SR_{a_p}}{SR_{a_q}}}{1 + \frac{log(\frac{T_{a_p}}{T_{a_q}})}{K_T}}.$$

SR bezeichnet die Erfolgsrate (Success Rate) und T die Zeit (Time). K_T ist ein Wert, der die relative Relevanz der Zeit festlegt. Da die Werte für K_T nicht sehr intuitive sind, wird in [SB00] vorgeschlagen, dass der Nutzer einen Wert von x% vorgibt. K_T errechnet sich dann aus $1/x\%$. Durch *x* kann der Nutzer vorgeben, wieviel Prozent der Erfolgsrate er gewillt ist zu opfern, um zehnmal schneller ein Ergebnis zu erhalten. Ist der Nutzer beispielsweise bereit, bis zu 10% der Erfolgsrate zu opfern, dann ist $K_T = 1/10\% = 10$. Würde er nur 1% angeben, dann wäre $K_T = 100$ und die Erfolgsrate würde höher gewichtet, da dem Nutzer ein genaueres Ergebnis offensichtlich wichtiger ist.
Wurden für eine Aufgabe drei Algorithmen $a_1,\ a_2,\ a_3$ vorausgewählt, dann berechnet sich der Rankwert für a_1 aus:

$$ARR_{a_1} = \frac{ARR_{a_1,a_2} + ARR_{a_1,a_3}}{2}.$$

Der Algorithmus mit dem höchsten Rankwert ist der geeignetste Algorithmus.

Vergleich BSIM und ARR: Stellt man beide Evaluierungsstrategien gegenüber, so stellt man fest, dass sie beide sowohl die Erfolgsrate als auch die Zeit einbeziehen und außerdem dem Nutzer die Möglichkeit geben, auf die Bestimmung des geeignetsten Verfahrens Einfluss zu nehmen.
Die BSIM-Strategie hat den Vorteil, dass sie einfach für weitere Evaluierungskriterien erweiterbar ist. Außerdem ist der Berechnungsaufwand zur Ermittlung der BSIM-Werte geringer als der zur Ermittlung der ARR-Werte, jedoch ergeben sich dort zwei Probleme:

1. Zum einen ist es für einen Nicht-Experten schwierig, absolute Grenzen für die Erfolgsrate und Zeit vorzugeben. Da der Meta-Lerner gerade für Nutzer gedacht ist, die kein Expertenwissen über die Klassifikationsalgorithmen besitzen, ist die Idee der Grenzwertvorgabe nicht praktikabel.

2. Zum anderen werden die einzelnen Differenzen nicht unterschiedlich gewichtet. Da die Zeit einen größeren Bereich möglicher Werte besitzt als die Erfolgsrate, dominiert die Zeit das Ergebnis.

Die ARR-Strategie ist vielleicht nicht so leicht verständlich, jedoch löst sie die Probleme der BSIM-Strategie.

1. Der Nutzer gibt anstelle von absoluten Werten für Erfolgsrate und Zeit einen relativen an, der aussagt, wieviel Prozent der Erfolgsrate er für ein zehnmal schnelleres Ergebnis zu opfern bereit ist.

2. Indem $log(\frac{T_{a_p}}{T_{a_q}})$ berechnet wird, minimiert man den Effekt, dass die Zeit das Ergebnis dominiert. Die Werte der Zeit schwanken dadurch um 1, wie es auch die Werte der Erfolgsrate tun.

Da es als wichtiger angesehen wird, dass eine Evaluierungsstrategie sinnvolle und praktikable Ergebnisse liefert, wird die ARR-Strategie gewählt. Die BSIM-Strategie mag zwar Vorteile betreffend der Effizienz und Erweiterbarkeit haben, dies wird jedoch als zweitrangig angesehen, wenn die Evaluierung an sich keine zufriedenstellenden Ergebnisse liefert.

Mit Hilfe der ARR-Evaluierungsstrategie kann die CB-Komponente nun den geeignetsten Algorithmus mit eventuellen Parameterbelegungen für Aufgaben bestimmen. Laut [BBG+99] ist es möglich, dass ein Fall mehrere Lösungen haben kann, denn manchmal ist es schwierig Attribute zu definieren, die Lösungen unterscheiden, die dicht beieinander liegen. Dies soll hier durch Einbeziehung der Interpretierbarkeit vermieden werden. Sollten sich die ARR-Werte der Algorithmen nicht wesentlich unterscheiden (Abweichung $< 5\%$), so wird derjenige Algorithmus ausgewählt, dessen Interpretierbarkeit am ehesten den Wünschen des Nutzers entspricht. Der Fallbasiseintrag ist mit der Aufgabenbeschreibung, den Nutzerpräferenzen und dem durch die Evaluierungsstrategie gefundenen Label nun vollständig und kann zur Fallbasis hinzugefügt werden.

4.4.4 Komponente zum Bestimmen des ähnlichsten Falles

Die Komponente zum Bestimmen des ähnlichsten Falles (SC-Komponente, Similar Case) stellt das Kernstück des Meta-Lerners dar. Sie ermittelt den der neuen Aufgabe ähnlichsten Fall aus der Fallbasis. Dazu fordert sie die Aufgabenbeschreibung der neuen Aufgabe von der TD-Komponente an und lässt die Skalentypen der Attribute bestimmen. Die Skalentypen vergleicht sie mit den von der AP-Komponente generierten Algorithmenprofilen, um diejenigen Algorithmen herauszufinden, die in der Lage sind, diese Aufgabe zu bearbeiten. Mit Hilfe der vorausgewählten Verfahren filtert sie die von der CB-Komponente aufgestellten Fallbasis. Die SC-Komponente ermittelt alle diejenigen Fälle, deren Lösung in der Menge der vorausgewählten Algorithmen enthalten ist. In Abbildung 10 werden diese Fälle als gefilterte, bisherige Erfahrungen bezeichnet.
Mit Hilfe der Aufgabenbeschreibung der neuen Aufgabe und den von der Eingabe entgegengenommenen Nutzerpräferenzen wird nun ein neuer Fall konstruiert, dessen Lösung noch unbekannt ist. Anhand dieser Meta-Attribute und dem Nearest-Neighbour-Verfahren wird der ähnlichste Fall bestimmt, dessen Lösung für die neue Aufgabe übernommen werden soll. Der ähnlichste Fall entspricht demjenigen mit der geringsten Distanz zum neuen Fall. Die Distanz *dist* zweier Fälle c_i und c_j berechnet sich aus:

$$dist(c_i, c_j) = \sum_x \delta(v_{x,c_i}, v_{x,c_j})$$

wobei v_{x,c_i} der Wert des Meta-Attributes x für c_i ist, und $\delta(v_{x,c_i}, v_{x,c_j})$ ist die Distanz zwischen den Werten des Meta-Attributes x für c_i und c_j. Um alle Meta-Attribute gleich zu gewichten, werden die Distanzen der einzelnen Meta-Attribute zuvor normalisiert.
Die Lösung des von der SC-Komponente ermittelten ähnlichsten Falles ist der Algorithmus, der für die neue Aufgabe empfohlen wird. Besitzt dieser Algorithmus keine Parameter, die die Performanz beeinflussen, so kann er direkt ausgegeben werden. Andernfalls wird das Ergebnis an die AA-Komponente weitergegeben, um die geeignesten Parameterbelegungen zu finden.

4.4.5 Komponente zur Adaptation

Die AA-Komponente (Adapt Algorithm) ist für die Anpassung der von der SC-Komponente gefundenen Lösung verantwortlich. Sie versucht, eine für die neue Aufgabe bestmögliche Parameterbelegung des Algorithmus' zu finden. Adaptations-Methoden werden genutzt, um die Lösung entsprechend zu modifizieren. In Abhängigkeit von sich unterscheidenden Meta-Attributbelegungen (neue Aufgabe vs. ähnlichste Aufgabe aus der Fallbasis) können unterschiedliche Kombinationen von Adaptations-Regeln angewandt werden. Die Adaptations-Regeln beschreiben die Parameteränderungen, die ausgedrückt werden durch Faktoren, Differenzen oder

qualitative Ausdrücke. In [SUD98] werden verschiedene Methoden für die Adaptation vorgestellt. Unter anderem wird die Shepard-Interpolation genannt, wenn es darum geht, dass mehrere metrisch skalierte Parameter adaptiert werden sollen. In Abhängigkeit davon, welche Parameterangaben in den Algorithmenprofilen gemacht worden, siehe Abschnitt 4.4.1, muss eine Adaptations-Methode gewählt werden.

Nachdem die Komponenten und ihre Fähigkeiten beschrieben wurden, sollen sie nun den Anforderungen aus Abschnitt 4.1 gegenübergestellt werden, um zu prüfen, ob alle Anforderungen erfüllt werden.

4.4.6 Vergleich der Architektur mit den Anforderungen

Um zu überprüfen, ob der konzipierte Meta-Lerner zur Auswahl von Klassifikationsverfahren, die Anforderungen aus Abschnitt 4.1 erfüllt, wird auf die Anforderungen einzeln und in der gleichen Reihenfolge wie in 4.1 eingegangen.

1. Es wurde gefordert, dass der Meta-Lerner Informationen über die im Data Mining Tool zur Verfügung stehenden Algorithmen haben muss:
   Mit Hilfe der AP-Komponente kann der Nutzer die Algorithmen registrieren. Automatisch werden dann Algorithmenprofile erstellt, die der Meta-Lerner nutzen kann, um den Suchraum der in Frage kommenden Verfahren zu bestimmen.

2. Um Meta-Lernen zu ermöglichen, wurde gefordert, dass der Meta-Lerner Aufgaben nutzen kann, für die bereits das geeignetste Verfahren entsprechend einer Evaluierungsstrategie ermittelt wurde:
   Die CB-Komponente baut eine Fallbasis auf, die Informationen über bereits bearbeitete Aufgaben enthält. Als Strategie zur Ermittlung des geeignetsten Verfahrens wird ein Ranking der Verfahren mit Hilfe der ARR-Methode erstellt. Auf die Fallbasis kann durch die SC-Komponente zugegriffen werden, um die darin enthaltenen Erfahrungen für die Auswahl eines geeigneten Verfahrens für eine neue Aufgabe nutzen zu können.

3. Damit die Aufgaben besser vergleichbar sind, wurde nach einer Möglichkeit der einheitlichen Aufgabenbeschreibung gesucht:
   Diese einheitliche Beschreibung wird durch Metadaten realisiert, die durch die TD-Komponente mit Hilfe eines Entscheidungsbaumes ermittelt werden.

4. Um eine für den Nutzer zufriedenstellende Auswahl treffen zu können, sollten Nutzerpräferenzen berücksichtigt werden:
   Da die Nutzerpräferenzen als Meta-Attribute in die Fallbasis mit aufgenommen werden und zusammen mit der neuen Aufgabe durch den Nutzer eingegeben werden können, werden sie sowohl bei der Bestimmung des ähnlichsten Falles als auch bei der Anpassung der Parameter berücksichtigt.

5. Damit die Verfahrensauswahl stattfinden kann, wurde ein Verfahren zum Meta-Lernen gefordert:
   Als Meta-Lern-Verfahren wird das Fallbasierte Schließen angewandt, das es erlaubt, aus der Menge der bereits bearbeiteten Aufgaben eine ähnliche zur neuen Aufgabe zu finden, um von dem dort angegebenen Algorithmus auf den gesuchten zu schließen. Die Suche nach dem ähnlichsten Fall, wird von der SC-Komponente ausgeführt. Mit Hilfe der AA-Komponente ist es sogar möglich, eventuelle Parameter entsprechend der neuen Aufgabe und den gegebenen Nutzerpräferenzen anzupassen.

6. Der Erfolg des Meta-Lerners sollte durch eine Evaluierungstrategie beurteilbar sein:
   Die SC-Komponente nutzt zur Ermittlung des ähnlichsten Falles das Nearest-Neighbour-Verfahren, welches die neue Aufgabe entsprechend ihrer Metadaten klassifiziert. Wie bei anderen Klassifikationsverfahren kann man auch in diesem Fall durch Testdaten, Kreuzvalidierung oder Bootstrap (siehe Abschnitt 3.3) die Fehler- bzw. Erfolgsrate ermitteln, deren Ausmaß Aufschluss über den Erfolg des Meta-Lerners gibt. Außerdem kann auch hier der Zeitfaktor berücksichtigt werden, darauf wird im nächsten Punkt eingegangen.

7. Es wurde gefordert, dass der Meta-Lerner effizient arbeitet:
   Wie effizient ein Meta-Lerner arbeitet, hängt sicherlich stark von seiner Implementierung und vom Lerner-Pool des Data Mining Tools ab. Trotzdem kann man die folgende allgemeine Überlegung anstellen: Man geht davon aus, dass die Algorithmenprofile erstellt sind und die Fallbasis bereits aufgebaut ist. So muss der Meta-Lerner, um eine Empfehlung ausgeben zu können vor allem zwei Schritte ausführen, die die Effizienz beeinflussen:

   (a) Mit Hilfe der TD-Komponente muss die Beschreibung der Aufgabe generiert werden, was man mit dem Ausführen des Entscheidungsbaum-Verfahren abschätzen kann. Die Ausführungszeit dieses Schrittes ist abhängig von der gegebenen Aufgabe.

   (b) Dann muss der ähnlichste Fall aus der Fallbasis bestimmt werden, was man mit dem Ausführen des Nearest-Neighbour-Verfahren abschätzen kann. Die Ausführungszeit dieses Schrittes ist abhängig von der Größe der Fallbasis.

   Demnach bestimmt das Ausführen zweier Klassifikationsverfahren den Aufwand, um eine Empfehlung ausgeben zu können. Eine grobe Abschätzung führt also zu dem Ergebnis, dass sich der Einsatz des Meta-Lerners lohnt, wenn aus einer Menge von mehr als zwei Klassifikationsalgorithmen ausgewählt wird.

8. Weiterhin wurde gefordert, dass der Meta-Lerner adaptive Fähigkeiten besitzt:

Aufgrund der Wahl des Fallbasierten Schließens als Meta-Lern-Verfahren ist es möglich, die bisherigen Erfahrungen, also die Einträge der Fallbasis, inkrementell zu erweitern. Dies erfolgt mit Hilfe der CB-Komonente. Änderungen im Lerner-Pool können mit Hilfe der AP-Komponente dem Meta-Lerner bekannt gemacht werden, jedoch muss zusätzlich die Fallbasis neu generiert werden.

Jede der gestellten Anforderungen wird durch den konzipierten Meta-Lerner erfüllt, d.h. diese Beschreibung kann als Grundlage genutzt werden, um einen Meta-Lerner zu implementieren. Die Umsetzung des Meta-Lerners zur Auswahl von Klassifikationsverfahren und dessen Ergebnisse werden in Kapitel 5 beschrieben.
Im nächsten Abschnitt geht man darauf ein, wie Meta-Lerner zur Auswahl von anderen Verfahren als den Klassifikationsverfahren aussehen könnten bzw. welche Unterschiede sie aufweisen.

4.5 Unterschiede zwischen Meta-Lernern für verschiedene Data-Mining-Aufgaben

Es wurde bereits in Abschnitt 4.4 erwähnt, dass sich Meta-Lerner zur Auswahl von Algorithmen unterschiedlicher Verfahrensfamilien unterscheiden. In diesem Abschnitt wird geklärt, welche Unterschiede gemeint sind.
Der Lernverfahrenauswahlprozess ist, wie er in Abbildung 8 dargestellt wird, für alle Verfahrensfamilien gleich. Auch die grundsätzlichen Aufgaben der daraus abgeleiteten Komponenten, siehe Abbildung 11, unterscheiden sich nicht. Die Unterschiede liegen im Detail der einzelnen Komponenten. Auf die Komponenten wird nun genauer eingegangen:

- **AP-Komponente:** Die AP-Komponente hat die Aufgabe Lerner-Profile zu erstellen. In Abschnitt 4.4.1 wurde darauf eingegangen, welche Einträge das Profil von Klassifikationsverfahren enthält. Angaben, wie mögliche Skalentypen der Attribute, Angaben zu Parametern und die Einschätzung der Interpretierbarkeit, sind sicherlich auch für die Beschreibung von Verfahren der Assoziation und des Clusterings denkbar. Inwieweit sie zur Vorauswahl von Algorithmen bzw. zur Bestimmung des geeignetsten Verfahren geeignet sind, ist fraglich und müsste näher untersucht werden.
  Der Eintrag über den Skalentyp des Klassenattributes ist jedoch typisch für Klassifikationsverfahren und ist zur Beschreibung von Algorithmen anderer Verfahrensfamilien ungeeignet. So ist es denkbar, dass die Algorithmen jeder Verfahrensfamilie spezielle Eigenschaften besitzen, die in die Profile aufgenommen werden sollten. Im Fall von Assoziationsalgorithmen kann die Eigenschaft, ob man Regelmuster, siehe [KMR+94], vorgeben kann, um den Suchraum für Assoziationsregeln einzuschränken, bedeutsam sein. Im

Gegensatz dazu könnte es bei der Lösung von Clustering-Aufgaben von Interesse sein, ob die Algorithmen eine Instanz genau einem Cluster oder mehreren zuordnen.
Es ist offensichtlich, dass sich die von der AP-Komponente generierten Algorithmen-Profile für unterschiedliche Verfahrensfamilien unterscheiden.

- **TD-Komponente:** Auch die von der TD-Komponente erzeugte Aufgabenbeschreibung unterscheidet sich für unterschiedliche Verfahrensfamilien. Allein die Art der Aufgabe erfordert es, dass unterschiedliche Aspekte berücksichtigt werden. Eine Klassifikationsaufgabe ist dadurch gekennzeichnet, dass die zu lernenden Klassen angegeben werden und Attribute, die der Findung eines Klassifikationsmodells dienen. Bei der Assoziation werden anstelle der zu lernenden Klassen die Kennzahlen Support und Konfidenz vorgegeben und beim Clustering die Anzahl der zu bildenden Cluster.
  Für den hier konzipierten Meta-Lerner wurde eine nur für Klassifikationsaufgaben geeignete Methode zur Aufgabenbeschreibung gewählt. Aber selbst wenn eine für alle Verfahrensfamilien einheitliche Strategie zur Aufgabenbeschreibung, beispielsweise der statistische und informationstheoretische Ansatz, gewählt werden würde, muss aus der Vielzahl der möglichen Kennzahlen, eine für die Verfahrensfamilie geeignete Teilmenge gefunden werden.

- **CB-Komponente:** Die CB-Komponente ist für den Aufbau der Fallbasis verantwortlich. Die Grundstruktur der zu generierenden Einträge, bestehend aus den Meta-Attributen auf der einen Seite und dem Klassenlabel auf der anderen Seite, ist bei allen Verfahrensfamilien gleich. Jedoch unterscheiden sich die Zusammensetzung der Meta-Attribute und die Evaluierungsstrategie zur Bestimmung des Klassenlabels.
  Die Meta-Attribute setzen sich aus den Metadaten der Aufgabenbeschreibung und den Nutzerpräferenzen zusammen. Auf die Unterschiede bei der Aufgabenbeschreibung wurden im vorherigen Punkt bereits eingegangen. Durch die Nutzerpräferenzen ist es dem Nutzer möglich, auf den Auswahlprozess Einfluss zu nehmen. Bei der Klassifikation kann er dies durch das Gewichten des Fehlerraten-Zeit-Verhältnisses und der Angabe der gewünschten Interpretierbarkeit bewirken. Jedoch ist die Fehlerrate als Metrik einer Evaluierungsstrategie für Assoziation bzw. Clustering nicht anwendbar. So muss eine jeweils geeignete Evaluierungsstrategie definiert werden, um den geeignetsten Assoziations- bzw. Clustering-Algorithmus für eine Aufgabe zu bestimmen. Dabei können dann entsprechende Nutzerpräferenzen berücksichtigt werden.

- **SC-Komponente:** Die SC-Komponente ermittelt denjenigen Fall aus der Fallbasis, der der gegebenen Aufgabe am ähnlichsten ist. Dazu wird die Distanz zwischen der neuen Aufgabe und den Einträgen der Fallbasis berechnet. Im Fall des in dieser Arbeit konzipierten Meta-Lerners wird zunächst

von einer einheitlichen Gewichtung der Meta-Attribute ausgegangen. Es ist aber durchaus denkbar, dass selbst wenn zur Aufgabenbeschreibung für unterschiedliche Data-Mining-Aufgaben gleiche Metadaten verwendet werden würden, diese unterschiedlich gewichtet werden müssten, um bestmögliche Ergebnisse zu erzielen.

- **AA-Komponente:** Die AA-Komponente dient der Anpassung der Parameter des gefundenen Algorithmus. Die Parameter der Algorithmen, egal welcher Verfahrensfamilie sie angehören, unterscheiden sich und müssen individuell angepasst werden. Die Adaptation erfolgt aber aufgrund von Differenzen zwischen den Meta-Attributen, also der Aufgabenbeschreibung und Nutzerpräferenzen. Diese unterscheiden sich, wie zuvor festgestellt für die einzelnen Verfahrensfamilien.

Die wesentlichen Unterschiede liegen demzufolge in dem Aufbau der Algorithmenprofilen, der Art der Aufgabenbeschreibung, den Nutzerpräferenzen und der Evaluierungsstrategie zum Bestimmen des geeignetsten Algorithmus'.

4.6 Zusammenfassung

Im ersten Teil dieses Kapitels wurden Anforderungen aufgestellt, die ein Meta-Lerner erfüllen muss, damit die Auswahl eines geeigneten Lernfahrens für eine Data-Mining-Aufgabe möglich ist. Anhand dieser Anforderungen wurden dann zunächst die Ein- und Ausgaben des Meta-Lerners bestimmt. Anschließend wurde der Auswahlprozess beschrieben und die zur Realisierung des Prozesses notwendigen Komponenten identifiziert. Weiterhin wurden die Vor- und Nachteile der Meta-Lern-Verfahren, Meta-Klassifizierer und Fallbasiertes Schließen, abgewogen und sich für die Methode des Fallbasierten Schließens entschieden.
In einem nächsten Schritt wurde ein Meta-Lerner speziell für die Auswahl von Klassifikationsverfahren konzipiert. Dazu wurden die zuvor identifizierten Komponenten konkretisiert. Es wurde auf die Algorithmenprofile zur Beschreibung der Klassifikationsverfahren eingegangen. Außerdem wurde diskutiert, welche Art der Aufgabenbeschreibung gewählt wird, wobei die Entscheidung auf den Entscheidungsbaum-basierten Ansatz fiel. Zur Bestimmung des geeignetsten Klassifikationsverfahrens entschied man sich aus zwei Alternativen für die ARR-Evaluierungsstrategie. Bei einem Vergleich des konzipierten Meta-Lerners mit den anfangs gestellten Anforderungen konnte festgestellt werden, dass der Meta-Lerner allen Anforderungen gerecht wird.
Abschließend wurde auf Unterschiede aufmerksam gemacht, die die Empfehlung von Algorithmen unterschiedlicher Verfahrensfamilien erfordern. Bedeutend ist vor allem, dass eine von der Data-Mining-Aufgabe abhängige Evaluierungsstrategie definiert werden muss und das zur Aufgabenbeschreibung unterschiedliche Meta-Daten genutzt werden. Die dargestellten Unterschiede bekräftigen die Annahme, dass spezialisierte Meta-Lerner für unterschiedliche Aufgabentypen des

Data Mining notwendig sind.
Im nächsten Kapitel wird beschrieben, inwieweit der hier konzipierte Meta-Lerner zur Auswahl von Klassifikationsverfahren umgesetzt wurde, und es werden die vom Meta-Lerner ausgegebenen Empfehlungen evaluiert.

5 Umsetzung des konzipierten Meta-Lerners

Nachdem im vorherigen Kapitel ein Meta-Lerner zur Auswahl von Klassifikationsverfahren konzipiert wurde, konzentriert sich dieses Kapitel auf die Umsetzung dieses Meta-Lerners. Der entstandene Prototyp trägt den Namen “Medam”. Im Abschnitt 3.2 wurde bereits darauf eingegangen, dass man die Aufgabe eines Meta-Lerners zur Verfahrensauswahl auch als Meta Data Mining bezeichnen kann. Medam wurde von **Me**ta **Da**ta **M**ining abgeleitet.
Im folgenden Abschnitt wird beschrieben, an welches Data Mining Tool der Meta-Lerner angebunden wird. Danach wird auf die zur Realisierung verwendeten Algorithmen eingegangen. Der entstandene Prototyp wird anschließend mit den Meta-Lernern aus Abschnitt 3.7 verglichen. Abschließend werden Tests zur Evaluierung von Medam beschrieben und die dabei erzielten Ergebnisse dargestellt.

5.1 Wahl des Data Mining Tools

Die Komponenten des Meta-Lerners sollten auf Basis eines existierenden Data Mining Tools implementiert werden. Zur Verfügung gestellt werden konnten der DB2 Intelligent Miner for Data von IBM (siehe [IBM99]) und Weka (siehe [WF01]), ein System, das an der Universität von Waikato in Neuseeland entwickelt wurde.

5.1.1 DB2 Intelligent Miner for Data

Das kommerzielle Produkt, Intelligent Miner for Data, von IBM ist sehr umfangreich. Neben den eigentlichen Mining-Algorithmen umfasst es Funktionen zur Vorverarbeitung von Daten und eine Komponente zur Visualisierung der Mining-Ergebnisse, siehe [IBM99]. Dem Nutzer werden in Version 6 ein Verfahren für Assoziationsaufgaben und zwei Verfahren für Aufgaben des Clustering zur Verfügung gestellt. Klassifikationsaufgaben können entweder durch ein Baumverfahren, ein Verfahren auf Basis eines Neuronalen Netzes oder mit Hilfe verschiedener Statistischer Verfahren bearbeitet werden. Die zu analysierenden Daten können in der Datenbank oder als Datei vorliegen. Durch API-Funktionen (Environment layer API, Result API) wird es dem Nutzer ermöglicht, das Ausführen von Verfahren zu kontrollieren und erzielte Ergebnisse weiterzuverarbeiten.

5.1.2 Weka

Weka ist ein überschaubares Data-Mining-Werkzeug, das für Waikato Environment for Knowledge Analysis steht. Es wurde auf der Basis von Java entwickelt, und der Quellcode ist frei verfügbar. Weka bietet ein Assoziationsverfahren, zwei Verfahren für das Clustering und mehrere Klassifikationsverfahren. Die Architektur von Weka erlaubt es, die Menge der Verfahren mit eigenen zu erweitern. Es stellt sogar

eine Testumgebung für Klassifikationsverfahren zur Verfügung, mit der die Algorithmen auf verschiedene Fähigkeiten untersucht werden können. Außerdem werden Methoden zur Vorverarbeitung der Daten und zur Evaluierung der Ergebnisse bereitgestellt. Die Daten müssen allerdings in einem speziellen ARFF-Format vorliegen, siehe [WF01].

Beide Syteme stellen eine Vielzahl von Klassifikationsverfahren zur Verfügung, so dass eine Auswahlempfehlung durch einen Meta-Lerner für beide sinnvoll ist. Durch die Nichtveröffentlichung der verwendeten Verfahren erschwert IBM jedoch die Erstellung von Algorithmenprofilen.
Für die Realisierung von Medam wurde das Weka-Tool ausgewählt. Dadurch, dass der Quellcode zur Verfügung steht, kann auf bestehende Methoden zurückgegriffen werden. So eignet sich beispielsweise die Testumgebung für das Erstellen von Algorithmenprofilen. Vorhandene Verfahren, wie Entscheidungsbaum-Verfahren und Nearest Neighbour, können für das Meta-Lernen genutzt und einfach erweitert werden. Die bereits implementierten Evaluierungs-Methoden erleichtern die Ermittlung der Fehlerrate und damit die Bestimmung des geeignetsten Verfahrens. Nachteilig ist lediglich die Erstellung des ARFF-Formats und eine fehlende Datenbankanbindung.

5.2 Umsetzung der Komponenten

Im folgenden geht es um die Umsetzung des konzipierten Meta-Lerners auf Basis des Weka-Tools. Als Programmiersprache wurde dazu Java gewählt, zum einen, weil Weka in Java geschrieben wurde, und zum anderen, weil Java sich auszeichnet durch Plattformunabhängigkeit, Objektorientierung und schnelles Prototyping. Java erlaubt es, Klassen als Packages zu organisieren. Weka besteht aus mehreren Packages, eines davon fasst beispielsweise alle Filterfunktionen zur Datenvorverarbeitung zusammen (`weka.filters`) und ein anderes enthält die Verfahren für das Clustering (`weka.clusterers`). Klassen zur Realisierung des Meta-Lerners wurden in einem neuen Package `weka.meta` zusammengefasst. Folgende bestehenden Klassen bzw. Packages des Weka-Tools wurden erweitert:

- `C45ModelSelection, ClassifierSplitModel, ClassifierTree, J48 (weka.classifiers.j48)`
- `CheckClassifier, Evaluation, IBk (weka.classifiers)`
- `FastVector (weka.core)`

Auf einige der Erweiterungen wird in den anschließenden Unterabschnitten eingegangen.

5.2.1 Komponente zum Generieren von Algorithmenprofilen

Das WEKA-Tool bietet die Möglichkeit sich mit dem Werkzeug `CheckClassifier` einen Überblick über die Eigenschaften von Klassifikationsalgorithmen zu verschaffen. Aus der Menge der Eigenschaften werden von `weka.meta.AlgorithmProfile` nur einige zur Beschreibung der Algorithmen genutzt. Ein Algorithmusprofil setzt sich aus den in Tabelle 5 angegebenen Attributen zusammen. Bis auf den Namen des Algorithmus', der als String angegeben wird, werden

Attribut	Ausprägung
`algorithmName`	`string`
`classNominal_predictorNominal`	{`true,false`}
`classNominal_predictorNumeric`	{`true,false`}
`classNumeric_predictorNominal`	{`true,false`}
`classNumeric_predictorNumeric`	{`true,false`}
`attributeString`	{`true,false`}

Tabelle 5: Attributvektor zur Beschreibung von Algorithmen

die Eigenschaften durch boolsche Attribute repräsentiert. Das Attribut `classNominal_predictorNominal` beispielsweise sagt aus, ob ein Klassifikationsverfahren nominale Attribute, einschließlich dem Klassenattribut, verarbeiten kann. Angaben über die Interpretierbarkeit und mögliche Parameter wurden in der ersten Version von Medam noch nicht in die Algorithmenprofile aufgenommen. Die Profile werden in der Datei `AlgorithmProfiles.arff` abgelegt.

5.2.2 Komponente zum Generieren von Aufgabenbeschreibungen

Mit dem Entscheidungsbaum-basierten Ansatz sollen die Klassifikationsaufgaben beschrieben werden. Das Weka-Tool stellt zwei Entscheidungsbaumverfahren zur Verfügung und zwar den Id3-Algorithmus und das C4.5-Entscheidungsbaumverfahren (`weka.classifiers.j48.J48`). Da der Id3-Algorithmus nur mit nominalen Attributen, aber J48 auch mit numerischen Attributen arbeiten kann (ausgenommen dem Klassenattribut), wird J48 zur Beschreibung genutzt. Klassen des j48-Package werden demzufolge um Methoden erweitert, die die in Abschnitt 4.4.2 erläuterten Metadaten aus dem Entscheidungsbaum generieren:

- `public double getNodesPerAttribute()`
- `public double getNodesPerInstance()`
- `public double getCorroboration()`
- `public double getAvgGainRatioDiff()`

- `public int getMaxDepth()`
- `public int getNumRepeatedNodes()`
- `public double getShape()`
- `public double getHomogeneity()`
- `public double getImbalance()`
- `public int getInternalSym()`

`weka.meta.TaskDescription` lässt den Entscheidungsbaum für die gegebenene Daten aufbauen und fasst die berechneten Kennzahlen zu Metadaten zusammen.

5.2.3 Komponente zum Aufbau der Fallbasis

Die bisherigen Erfahrungen werden in der Datei `CaseBase.arff` abgelegt. Ein Eintrag setzt sich aus den Metadaten der Aufgabe, den Nutzerpräferenzen und dem geeignetsten Algorithmus zusammen. Als Nutzerpräferenz wird in der ersten Version von Medam nur `timeRelevance` aufgenommen, also die Angabe, wie stark die Zeit gegenüber der Fehlerrate bei der Evaluierung gewichtet werden soll. Bisher wurden drei Einträge pro Datensatz in die Fallbasis aufgenommen, die sich jeweils in `timeRelevance` unterscheiden. Laut [SB00] scheinen dafür die Werte 10%, 1% (default) und 0.1% sinnvoll (K_T=10, 100 und 1000).
Für das Generieren der Fälle ist die Klasse `weka.meta.CaseBase` verantwortlich. Sie erzeugt pro Aufgabe drei Objekte der Klasse `ClassificationTask`. Wie für das Erzeugen eines Eintrages vorgegangen wird, zeigt Abbildung 14. Um

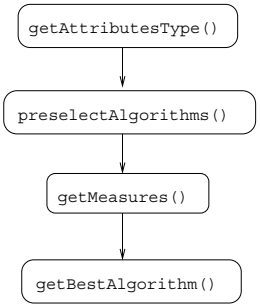


Abbildung 14: Generieren eines Falles

den geeignetsten Algorithmus für die jeweilige Aufgabe zu bestimmen, müssen die Fehlerraten und Zeiten für die vorausgewählten Algorithmen ermittelt werden (`getMeasures()`). Die Fehlerrate wird mit Hilfe der Klasse `Evaluation`

durch zehnfache Kreuzvalidierung (siehe Abschnitt 3.3) ermittelt. Die benötigte Zeit wird für alle zehn Durchläufe durch die von Java zur Verfügung gestellten Methode `System.currentTimeMillis()` bestimmt und anschließend gemittelt. Danach werden die ARR-Werte berechnet, `getBestAlgorithm()` liefert dann den Algorithmus mit dem höchsten Rankwert, um den Fallbasiseintrag zu vervollständigen.

5.2.4 Komponente zum Bestimmen des ähnlichsten Falles

Das von Weka bereitgestellte Verfahren `weka.classifiers.IBk` wird genutzt, um den ähnlichsten Fall zur gegebenen Aufgabe zu bestimmen. Zunächst wird ein Objekt der Klasse `ClassificationTask` erzeugt und die Metadaten mit `getMetaData` angefordert. Auf die Einträge der Datei CaseBase.arff wird das kNN-Verfahren angewandt (k=1), es werden also die Distanzen zur neuen Aufgabe ermittelt. Dabei werden die Meta-Attribute zunächst alle gleich gewichtet. Das Label des ähnlichsten Falles wird als Empfehlung ausgegeben.

Die Komponente zur Anpassung der Parameter wurde in der ersten Version von Medam nicht umgesetzt.

Möchte der Nutzer ein Verfahren für eine Aufgabe empfohlen bekommen, so ruft er von der Kommandozeile folgendes auf:

```
weka.meta.Medam [-M <Mining Task>] -t <file> [-c <index>]
[-u <timeRelevance>]
```

- `<Mining Task>`: Wahl der Data-Mining-Aufgabe. Die möglichen Optionen werden weiter unten angegeben.
- `<file>`: Pfad, unter dem die zur Aufgabe gehörenden Daten zu finden sind.
- `<index>`: Index des Klassenattributes. Wird `-c <index>` nicht angegeben, so wird das letzte Attribut aus `<file>` als Klassenattribut angenommen.
- `<timeRelevance>`: Gewichtung des Fehlerraten-Zeit-Verhältnisses. Empfohlen wird ein Wert zwischen 10 und 0.1. Wird `-u <timeRelevance>` nicht angegeben, dann wird der Wert auf eins gesetzt (K_T=100).

Da Medam vielleicht in der Zukunft erweitert wird, so dass er auch Algorithmen für andere Data-Mining-Aufgaben empfehlen kann, wurde das Flag `-M <Mining Task>`, eingeführt. Vorgegeben sind bisher für <Mining Task>:

<Mining Task>	Bedeutung
A	Association
U	Unsupervised Classification/ Clustering
S	Supervised Classification (default)

Die überwachte Klassifikation ist die voreingestellte Option.
Ausgegeben wird dann der Name des empfohlenen Algorithmus, beispielsweise `weka.classifiers.NaiveBayes`.

Im folgenden Abschnitt werden die Unterschiede zwischen dem Meta-Lerner Medam und den in Abschnitt 3.7 beschriebenen Meta-Lernern dargestellt.

5.3 Vergleich des prototypisch implementierten Meta-Lerners Medam mit denen aus Abschnitt 3.7

In diesem Abschnitt wird der Prototyp Medam mit den Meta-Lernern ENTRENCHER, AST und NOEMON verglichen.
Allen gemeinsam ist, dass sie Meta-Lerner zur Auswahl von Klassifikationsaufgaben sind. Sie verwenden bereits bearbeitete Aufgaben, um eine Zuordnung von Aufgaben zu Klassifikationsverfahren zu finden. Die Klassifikationsverfahren werden bei allen nur mit den Default-Einstellungen ihrer Parameter betrachtet.
Jedoch unterscheiden sie sich in der Art, wie sie die Aufgaben beschreiben und welches Meta-Lern-Verfahren sie einsetzen. So nutzen der hier konzipierte Meta-Lerner und der ENTRENCHER Metadaten, die anhand eines Entscheidungsbaumes berechnet werden, während AST und NOEMON statistische und informationstheoretische Kennzahlen verwenden. Bis auf den ENTRENCHER, der einen Meta-Klassifizierer als Meta-Lern-Verfahren einsetzt, nutzen die anderen die Methode des Fallbasierten Schließens. Weiterhin kann gesagt werden, dass nur Medam außer der Fehlerrate noch die Zeit als Evaluierungskriterium zur Ermittlung des geeignesten Verfahren berücksichtigt. Algorithmenprofile zur Beschreibung der Klassifikationsverfahren werden von AST und Medam erstellt. Abschließend lässt sich feststellen, dass nur AST und Medam es dem Nutzer ermöglichen, Einfluss auf die Auswahl des geeignetsten Verfahren zu nehmen. Die Tabelle 6 fasst das Ergebnis von der Gegenüberstellung der vier Meta-Lerner noch einmal zusammen.

5.4 Evaluierung

Um die Qualität der von Medam ausgegebenen Empfehlungen bewerten zu können, wurden Experimente durchgeführt. Ihre Ergebnisse werden in diesem Abschnitt beschrieben.

	Medam	ENTRENCHER	AST	NOEMON
Art der Aufgaben-beschreibung	Entschei-dungsbaum basiert	Entscheidungs-baum basiert	statistische und informations-theoretisch	statistische und informations-theoretisch
Meta-Lern-Verfahren	Fallbasiertes Schließen	Meta-Klassifizierer	Fallbasiertes Schließen	Fallbasiertes Schließen
Evaluierungs-kriterien	Fehlerrate, Zeit	Fehlerrate	Fehlerrate	Fehlerrate
Algorithmen-profile	√	-	√	-
Nutzer-präferenzen	√	-	√	-

Tabelle 6: Vergleich der Meta-Lerner

5.4.1 Verwendete Daten

Die für die Experimente verwendeten Daten wurden aus [MA94] entnommen. Zuvor wurde bereits erwähnt, dass es schwierig ist, aufgrund des Datenschutzes und des Betriebsgeheimnisses, ausreichend Datensätze zu bekommen. In [MA94] versucht man, eine Sammlung von frei zur Verfügung stehenden Data-Mining-Aufgaben aufzubauen. 19 der dort bereitgestellten Klassifikationsaufgaben wurden bisher für den Aufbau der Fallbasis genutzt. Einen Überblick über die verwendeten Datensätze ist im Anhang C zu finden. Da für jeden Datensatz, jeweils mit unterschiedlicher Gewichtung des Fehlerraten-Zeit-Verhältnisses, dreimal das geeignetste Verfahren ermittelt wird, enthält die Fallbasis insgesamt 57 Einträge.

5.4.2 Verwendete Algorithmen

Registriert wurden zunächst neun der Klassifikationsverfahren, die Weka zur Verfügung stellt. Tabelle 7 gibt einen Überblick über die verwendeten Verfahren.
Bald stellte sich heraus, dass aufgrund der Vorauswahl der Algorithmen und der gegebenen Datensätzen immer sieben der Algorithmen übrigblieben. Keiner der 19 Datensätze konnte durch LinearRegression oder LWR bearbeitet werden, weil die Klassenattribute der Datensätze ausschließlich nominal sind. Deshalb werden in den Ergebnistabellen von Anhang D immer nur die gleichen sieben Algorithmen aufgelistet.

5.4.3 Darstellung der Ergebnisse

Die Ergebnistabelle 8 des *house-votes-84*-Datensatzes ist in diesem Abschnitt angegeben, um die Darstellung der Ergebnisse zu erläutern.

Verfahren	Klasse
Mehrheits-/Durchschnitts-vorhersagesystem	`weka.classifiers.ZeroR`
1R	`weka.classifiers.OneR`
NaiveBayes	`weka.classifiers.NaiveBayes`
Instanzbasiertes Lernsystem	`weka.classifiers.IB1`
C4.5	`weka.classifiers.j48.J48`
PART-Regellernsystem	`weka.classifiers.j48.PART`
Entscheidungsbäume mit einer Ebene	`weka.classifiers.DecisionStump`
Lineare Regression	`weka.classifiers.LinearRegression`
Lokal gewichtete Regression	`weka.classifiers.LWR`

Tabelle 7: Registrierte Verfahren

In der ersten Zeile sind die drei Werte angegeben, die für `timeRelevance`, also der Fehlerraten-Zeit-Gewichtung, zum Aufbau der Fallbasis gewählt wurden. Darunter sind die errechneten ARR-Rankwerte für die sieben vorausgewählten Algorithmen aufgelistet. Die Werte wurden bereits absteigend sortiert, so dass der für den Datensatz geeignetste Algorithmus ganz oben steht. In der Fallbasis sind dem-

`timeRelevance`	`10`		`1`		`0.1`	
1	OneR:	1.5	OneR:	1.35	J48:	1.35
2	DS:	1.49	J48:	1.34	**PART:**	1.34
3	**NB:**	1.38	DS:	1.34	OneR:	1.33
4	J48:	1.2	**PART:**	1.32	DS:	1.33
5	PART:	1.14	IB1:	1.24	IB1:	1.27
6	IB1:	1.04	NB:	1.24	NB:	1.23
7	ZeroR:	1.04	ZeroR:	0.79	ZeroR:	0.77
CBR-k=1		ZeroR		PART*		PART*
CBR-k=7		**NB**		**PART***		**PART***
Z_2(ARR)		NB		IB1		PART*
C4.5		J48		J48*		J48*

Tabelle 8: Ergebnistabelle vom Datensatz *house-votes-84*

zufolge für den Datensatz *house-votes-84* diese drei Einträge zu finden:
<Metadaten>,10,weka.classifiers.OneR
<Metadaten>,1,weka.classifiers.OneR
<Metadaten>,0.1,weka.classifiers.j48.J48

In den anschließenden Zeilen der Ergebnistabelle sind die Ergebnisse der Experimente aufgelistet. Die besten Ergebnisse wurden bisher durch das Experiment CBR-k=7 erzielt, deshalb wurden die jeweiligen Algorithmen fett hervorgehoben. Auf die Bedeutung der Sterne, die an manchen Algorithmen zu finden sind, wird später eingegangen.

Vergleicht man die Rankings der Datensätze miteinander, so kann bereits folgendes festgestellt werden: Von den sieben vorausgewählten Algorithmen gibt es keinen Algorithmus, der für alle Datensätze immer ein sehr gutes Ranking erzielt. Ein von den Aufgaben abhängiges Empfehlen von Algorithmen ist demnach sinnvoll.
Außerdem kann man eine zweite Beobachtung bei einigen der Datensätzen machen. Der Algorithmus ZeroR ist ein recht einfacher Algorithmus und wenn sich seine Platzierung für einen Datensatz ändert, dann erfolgt das bei zunehmender Gewichtung der Erfolgsrate absteigend. Andere Algorithmen, wie beispielsweise der PART-Algorithmus, generieren ein besseres Klassifikationsmodell, benötigen dafür allerdings etwas mehr Zeit. Bei solchen Algorithmen kann man oftmals eine bessere Platzierung beobachten, wenn die Erfolgsrate bei der Evaluierung stärker bewertet wird. Die Ergebnistabelle 19 des Datensatzes *sick* verdeutlicht das sehr schön.

5.4.4 Experiment: CBR-k=1

In dem ersten Experiment wurde nur der ähnlichste Fall aus der Fallbasis bestimmt, d.h. der Parameter k, der die Anzahl der zu findenden ähnlichen Fälle für das Nearest-Neighbour-Verfahren festlegt, wurde auf eins gesetzt (k=1).
Nun wurden die Datensätze einzeln betrachtet (leave-one-out). Als erstes entfernte man die drei Fallbasiseinträge, die aufgrund dieses Datensatzes erzeugt wurden, aus der Fallbasis. Dann ließ man sich drei Empfehlungen vom Meta-Lerner mit der modifizierten Fallbasis für diesen Datensatz mit `timeRelevance`= 10%, 1% und 0.1% ausgeben. Das Ergebnis ist für jeden Datensatz in der Zeile CBR-k=1 seiner Ergebnistabelle zu finden.
Wertet man das Gesamtergebnis aus, stellt man fest, dass nur 19 der 57 Ausgaben (33%) dem geeignetsten Verfahren entsprechen. Betrachtet man jedoch die ARR-Werte genauer, so ist feststellbar, dass sie sich oftmals nicht signifikant unterscheiden. Lässt man eine 5%ige Abweichung vom besten Rankwert zu und zählt dann alle in diesem Bereich liegenden Algorithmen auch zu den geeignetsten, dann erhöht sich die Erfolgsrate auf 53%. Algorithmen, deren Rankwerte in dem 5%-Bereich liegen werden durch einen Stern gekennzeichnet.

Dieses immer noch inakzeptable Ergebnis von 53% kann verschiedene Ursachen haben:

1. Der gefundene ähnlichste Fall ist ein Ausreißer. In diesem Fall wäre es angebracht, mehrere zur gegebenen Aufgabe ähnliche Fallbasiseinträge zu be-

trachten.

2. Einige der Meta-Attribute sind weniger relevant für die Zuordnung des geeignetsten Verfahrens als andere. In diesem Fall, wäre eine Gewichtung der Meta-Attribute sinnvoll.

3. Die in der Fallbasis enthaltenen Fälle sind nicht repräsentativ genug. Es könnten auch redundante Fälle enthalten sein, die das Ergebnis negativ beeinflussen. Die Relevanz der einzelnen Fallbasiseinträge muss in diesem Fall überprüft werden.

In den folgenden Experimenten wird versucht, den ersten beiden Ursachen nachzugehen.

5.4.5 Experiment: CBR-k=7

Bei der Durchführung dieses Experimentes wurde ähnlich, wie in dem zuvor beschriebenen Experiment vorgegangen. Der Parameter k des Nearest-Neighbour-Verfahrens wurde jedoch erhöht. Die 5, 7 und 10 ähnlichsten Fälle zur gegebenen Aufgabe wurden betrachtet. Die Tabelle 9 zeigt die zehn ähnlichsten Fälle aus der Fallbasis zum Datensatz *labor* bei `timeRelevance`= 10.
Die Ergebnisse für k=5, 7 und 10 unterscheiden sich nicht wesentlich, nur in Ein-

Distanz	Datensatz	`timeRelevance`	Algorithmus
0.3459	contact-lenses	10	J48
0.3491	zoo	10	NB
0.3664	weather	10	IB1
0.4548	heart	10	NB
0.5844	iris	10	NB
0.6553	ionosphere	10	ZeroR
0.6781	wine	10	NB
0.7129	house-votes-84	10	OneR
1.0241	glass	10	OneR
1.0558	allhyper	10	ZeroR

Tabelle 9: Die zehn ähnlichsten Fälle zum Datensatz *labor* bei `timeRelevance` = 10

zelfällen liefert k=7 ein besseres Ergebnis und deshalb werden die Ergebnisse für k=7 in den Ergebnistabellen dargestellt.
Durch das Hinzuziehen mehrerer ähnlicher Fälle wurden beispielsweise die Empfehlungen für den Datensatz *labor* deutlich verbessert, siehe Tabelle 25. Für `timeRelevance` = 10 wurde erst der Algorithmus J48 empfohlen, der auf dem sechsten Platz lag. Durch das Betrachten der 7 ähnlichsten Datensätze wird nun

der geeignetste Algorithmus NB empfohlen.
Insgesamt konnte die Erfolgsrate dadurch auf 67% gesteigert werden.

5.4.6 Experiment: Z_2(ARR)

Das nächste Experiment dient dazu, die Ergebnisse von Medam mit denen einer anderen Vorgehensweise zu vergleichen. Medam verwendet die gleiche Evaluierungsstrategie wie Zoomed Ranking, deshalb ist es interessant herauszufinden, ob die unterschiedliche Vorgehensweise der Verfahren zu unterschiedlichen Ergebnissen führen trotz gleicher Evaluierungsstrategie.
Das in [SB00] beschriebene Zoomed Ranking verwendet statistische und informationstheoretische Kennzahlen zur Beschreibung der Klassifikationsaufgaben, in diesem Experiment werden die Entscheidungsbaum-basierten stattdessen genutzt. Außerdem gibt das Verfahren ein Ranking als Ergebnis aus und nicht ein einzelnes Verfahren. Die allgemeine Vorgehensweise des Zoomed Ranking ist die folgende:

1. Finde die k-ähnlichsten Datensätze zur gegebenen Aufgabe, nutze dazu aber nur die Meta-Attribute die der Aufgabenbeschreibung dienen, also nicht die Nutzerpräferenzen.
2. Für jeden dieser Datensätze fordere die Zeiten und Fehlerraten der registrierten Algorithmen an.
3. Errechne die $ARR^{d_i}_{a_p,a_q}$-Werte für jeden der k Datensätze, die hier mit d_i bezeichnet wurden.
4. Ermittle ein Ranking durch

$$ARR_{a_p,a_q} = \frac{\left(\sum_i ARR^{d_i}_{a_p,a_q}\right)}{k}$$

   und gibt es als Ergebnis aus.

Der einzige Unterschied bei der Evaluierung ist, dass das Ranking durch mehrere Datensätze beeinflusst wird. Bei Medam wird immer nur ein Datensatz betrachtet, siehe Abschnitt 4.4.3. In [SB00] wird festgestellt, dass Zoomed Ranking bei k=2 die robusteren Ergebnisse für unterschiedliche `timeRelevance`-Werte liefert, deshalb wird hier auch k=2 gesetzt. Als Ergebnis wird der Algorithmus mit dem höchsten Rankwert in die Ergebnistabellen, Zeile Z_2(ARR), aufgenommen.
Die geringere Erfolgsrate von 51% zeigt, dass die veränderte Vorgehensweise keinen positiven Einfluss auf das Ergebnis hat.

Der ENTRENCHER, siehe Abschnitt 3.7.1, verwendet auch Entscheidungsbaum-basierte Metadaten zur Aufgabenbeschreibung. Bei der Anwendung des C4.5-Algorithmus als Meta-Klassifizierer erreichte er allerdings eine durchschnittliche Fehlerrate von 84% bei boolschen Experimenten. Das bessere Ergebnis weißt darauf

hin, dass nicht alle der Meta-Attribute gleich relevant sind für die Auswahl des geeignetsten Verfahrens. Das nächste Experiment beschäftigt sich mit der Relevanz der Meta-Attribute.

5.4.7 Experiment: C4.5

Um festzustellen, ob das unterschiedliche Gewichten der Meta-Attribute eine Verbesserung der bisher erzielten Fehlerrate bringt, könnte man systematisch die verschiedenen Möglichkeiten durchprobieren. Das ist allerdings sehr zeitaufwendig. Einfacher ist es, die Fallbasiseinträge als Eingabe für ein Entscheidungsbaumverfahren zu nutzen. Das Verfahren ermittelt den Informationsgewinn der Attribute und führt dementsprechend die Verzweigung des Baumes durch. Die Idee ist nun, die Meta-Attribute gemäß ihrer Verwendung im Baum zu gewichten.
Die Anwendung des `weka.classifiers.j48.J48`-Algorithmus' auf die Fallbasis ergab den in Abbildung 15 dargestellten geprunten Entscheidungsbaum. Die

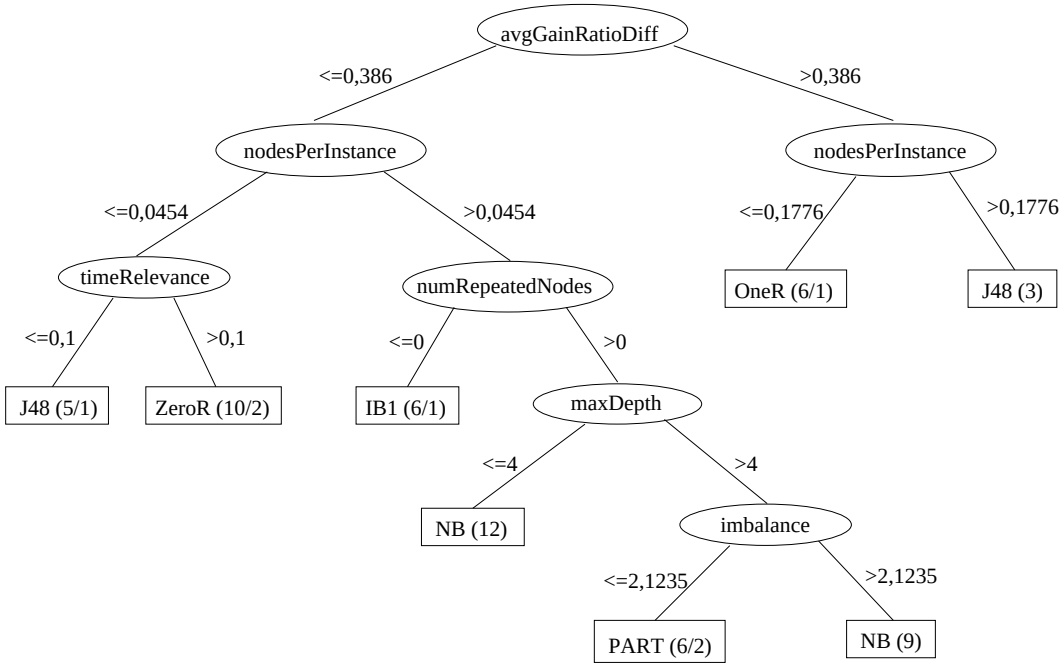


Abbildung 15: Entscheidungsbaum der Meta-Attribute

dabei erzielte und überraschend gute Erfolgsrate von 77% (stratefizierte zehnfache Kreuzvalidierung) kann jedoch nicht mit den zuvor erzielten Erfolgsraten verglichen werden. Bisher wurden immer alle drei Einträge zu einem Datensatz aus der Fallbasis entfernt, hier ist das zunächst nicht der Fall. Es ist also sehr wahrscheinlich, dass Einträge eines Datensatzes, die sich nur in dem Wert von `time-`

`Relevance` unterscheiden, sowohl in den Trainingsdaten als auch in den Testdaten enthalten sind. Wenn sich stark ähnelnde Beispiele in Trainings- und Testdaten befinden, begünstigt das eine bessere Erfolgsrate. Das wird bestätigt durch die 60%ige Erfolgsrate, die bei Anwendung des `weka.classifiers.IBk` auf die Fallbasis durch stratefizierte zehnfache Kreuzvalidierung erzielt wurde, wobei sämtliche Einträge eines Datensatzes, wie bei Anwendung des J48-Algorithmus', in der Fallbasis verblieben. Das Ergebnis von 60% ist nicht vergleichbar mit den zuvor erzielten 33% im Experiment CBR-k=1.
Um die Ergebnisse besser vergleichen zu können, wurde schließlich die Erfolgsrate ermittelt, die bei Anwendung des J48-Algorithmus' erzielt wird, wenn die Einträge nicht in der Fallbasis verbleiben. Tabelle 10 stellt die Werte gegenüber.

Verfahren	leave-one-out	leave-one-out + 5% Abweichung	stratefizierte zehnfache Kreuzvalidierung
IBk (k=1)	33%	53%	60%
C4.5	36%	54%	77%

Tabelle 10: Erfolgsraten

Die Frage ist nun, wie die Meta-Attribute gemäß ihrer Verwendung im erzeugten Entscheidungsbaum, siehe Abbildung 15, gewichtet werden. Da nicht alle Meta-Attribute als Split-Attribute verwendet wurden, ist es offensichtlich, dass nur diejenigen Attribute höher gewichtet werden, die im Entscheidungsbaum erscheinen. Wie stark man ein Attribut gewichtet, wird nun von der Tiefe des Knotens abhängig gemacht, in dem es verwendet wurde. Der Baum hat eine Gesamttiefe von 6. Da für das Attribut im Wurzelknoten der höchste Informationsgewinn ermittelt wurde, wird es auch am stärksten gewichtet, d.h. die Knotentiefe wird den Attributen absteigend als Gewicht zugeordnet. Tabelle 11 zeigt die Gewichtung der Meta-Attribute entsprechend ihrer Verwendung im Entscheidungsbaum. Wendet man den IBk-Algorithmus auf die Datensätze mit der unterschiedlichen Gewichtung der Meta-Attribute an, so erhöht sich die Erfolgsrate von 60% auf 73% (stratefizierte zehnfache Kreuzvalidierung). Die Steigerung der Erfolgsrate zeigt, dass die Meta-Attribute unterschiedlichen Einfluss auf die Auswahl des geeignetsten Verfahrens haben.

Im Verlauf der Experimente konnte die Erfolgsrate durch Veränderungen bereits deutlich gesteigert werden. Diese Veränderungen bezogen sich bisher auf die Anzahl der betrachteten ähnlichsten Fälle und der Gewichtung der Attribute. Vergleicht man beispielsweise die in Experiment CBR-k=7 bereits erzielte Erfolgsrate von 70% mit einer zufälligen Auswahl des geeignetsten Verfahrens, so ist das Ergebnis des Meta-Lerners wesentlich besser. Bei neun verwendeten Algorithmen würde man mit einer Wahrscheinlichkeit von nur 11% den geeignetsten wählen.

Attribut	Gewichtung
nodesPerAttribute	1
nodesPerInstance	5
avgLeafCorroboration	1
avgGainRatioDiff	6
maxDepth	3
numRepeatedNodes	4
shape	1
homogeneity	1
imbalance	2
internalSymmetry	1
timeRelevance	4

Tabelle 11: Gewichtung der Meta-Attribut

Sicherlich ist eine erneute Steigerung durch weitere Experimente möglich. Diese Experimente könnten zum einen die verwendeten Meta-Attribute genauer auf ihre Relevanz für die Auswahl des geeignetsten Verfahrens untersuchen und zum anderen Untersuchungen zur Relevanz von Fallbasiseinträgen durchführen.

5.5 Zusammenfassung

In diesem Kapitel wurde die Umsetzung des in Kapitel 4 konzipierten Meta-Lerners beschrieben. Das Data-Mining-Werkzeug Weka wurde als Basis für den entstehenden Prototypen namens Medam ausgewählt. Es wurde darauf eingegangen, wie Weka um die notwendigen Komponenten erweitert wurde, und wie Methoden, die vom Weka-Tool bereitgestellt werden, für die Implementierung wiederverwendet werden konnten. Anschließend wurde Medam anderen Meta-Lernern gegenübergestellt.
Im letzten Teil dieses Kapitels wurden vier Experimente zur Evaluierung des Meta-Lerners beschrieben:

1. Ermittlung der Erfolgsrate bei Betrachtung des ähnlichsten Falles,
2. Ermittlung der Erfolgsrate bei Betrachtung mehrerer ähnlicher Fälle,
3. Vergleich des Ergebnisses von Medam mit dem Ergebnis einer Zoomed-Ranking-ähnlichen Vorgehensweise,
4. Ermittlung der Erfolgsrate bei unterschiedlicher Gewichtung der Meta-Attribute.

Es konnte eine Verbesserung der Erfolgsrate um 14% erzielt werden, als anstelle des ähnlichsten Falles mehrerer ähnliche Fälle betrachtet wurden. Im Gegensatz

dazu hat die Zoomed-Ranking-ähnliche Vorgehensweise zu keiner Verbesserung der Erfolgsrate geführt. Desweiteren konnte durch das letzte Experiment gezeigt werden, dass nicht alle der Meta-Attribute die gleiche Relevanz bei der Bestimmung des geeignetsten Verfahren für eine neue Aufgabe haben. Insgesamt liefert Medam ein zufriedenstellendes Ergebnis.

6 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde ein Meta-Lerner konzipiert, der die Auswahl von Data-Mining-Verfahren steuert. Um die Vielfalt der Data-Mining-Verfahren und damit die Notwendigkeit einer automatisierten Auswahlsteuerung darzustellen, wurden zunächst einige der gebräuchlichsten Data-Mining-Verfahren vorgestellt und die durch diese Verfahren lösbaren Problemstellungen beschrieben.
Ausgangspunkt für die Entwicklung eines intelligenten Systems zur Auswahlsteuerung war die Idee des Meta-Lernens. Der Vorteil von Lernsystemen besteht darin, dass sie durch Beobachtung der Data-Mining-Verfahren automatisch lernen, in welchen Situationen welche Algorithmen anzuwenden sind. Dazu wurde der prinzipielle Aufbau eines Meta-Lerners bestimmt, der den allgemeinen Anforderungen an Meta-Lernern gerecht wird. Während sich eine Meta-Lerner-Architektur allgemein für alle Aufgabenarten des Data Mining, wie beispielsweise die Klassifikation, die Assoziation und das Clustering, angeben lässt, erfordert die Anwendbarkeit des Meta-Lerners eine Aufgaben-spezifische Instanziierung der eingebundenen Komponenten. Bedeutend ist vor allem, dass eine von der Data-Mining-Aufgabe abhängige Evaluierungsstrategie definiert werden muss und das zur Aufgabenbeschreibung unterschiedliche Meta-Daten, wie beispielsweise Entscheidungsbaum-basierte oder statistische und informationstheoretische, genutzt werden.
Weiterhin konnte festgestellt werden, dass der Erfolg eines Meta-Lerners abhängt vom eingesetzten Lernverfahren, von der Beschreibung der Problemstellung und den zur Auswahl stehenden Verfahren. Als Lernverfahren des Meta-Lerners wurde die Methode des Fallbasierten Schließens gewählt. Der Vorteil fallbasierter Verfahren liegt darin, dass sie besonders für kleine Datenmengen geeignet sind. Außerdem können durch die Lösungsanpassung gegebenenfalls Parameter adaptiert werden.
Da man der Klassifikation, verglichen mit anderen Verfahrensfamilien, überdurchschnittlich viele Algorithmen zuordnen kann, wurde die allgemeine Architektur für die Klassifikation konkretisiert. Algorithmenprofile, die Angaben über die Fähigkeit der Verfahren enthalten, dienen dem Einschränken des Suchraums potentiell geeigneter Verfahren. Als Evaluierungskriterien für die Klassifikationsverfahren wurden die Erfolgsrate und die Zeit genutzt. Dem Nutzer ist es möglich, mittels Gewichtung der Kriterien Einfluss auf die Verfahrensauswahl zu nehmen. Zur Beschreibung der Klassifikationsaufgaben wurde der neuere Ansatz von Bensusan auf der Basis von Entscheidungsbäumen gewählt. Entscheidungsbaumverfahren, die selbst den Klassifikationsverfahren zuzuordnen sind, eignen sich besonders gut, da sie implizite Eigenschaften von Klassifikationsaufgaben hervorbringen und Attribute unterschiedlichen Skalenniveaus gleichermaßen beschreiben.
Bausteine des konzipierten Meta-Lerners wurden auf Basis des Weka-Tools implementiert. Experimente mit dem entstandenen Prototypen Medam führten zur sukzessiven Verbesserung des Systems. Dies konnte beispielsweise durch das Gewichten der Attribute entsprechend ihres Informationsgewinns erreicht werden. Durch

Aufdecken von Redundanzen in der Fallbasis und durch die Integration von Heuristiken zur Bestimmung weiterer Stützpunkte ist eine weitere Steigerung der Erfolgsrate denkbar.
Abgesehen davon ist eine Datenbankanbindung erstrebenswert, um beispielsweise auf Fallbasiseinträge effizienter zugreifen zu können. Auch die Algorithmenprofile und die Performanzwerte (Zeit, Erfolgsrate) der Algorithmen, die zur Bestimmung des Labels eines Fallbasiseintragen ermittelt werden, sollten in einer Datenbank verwaltet werden. Änderungen im Lerner-Pool können dann effizienter umgesetzt werden.
Insgesamt ist der gewählte Ansatz als vielversprechend einzuschätzen. Interessant wäre eine tiefergehende Studie, die die Performanz der verschiedenen Meta-Lerner miteinander vergleicht. Voraussetzung für einen derartigen Vergleich ist, dass die Meta-Lerner die gleichen potentiell geeigneten Algorithmen zur Auswahl haben und die bisherigen Erfahrungen auf denselben Datensätzen beruhen. Die Ergebnisse einer solchen Studie könnten Klarheit über die Eignung von Aufgabenbeschreibungsstrategien und Meta-Lern-Verfahren bringen. Das Weka-Tool und die UCI-Daten bieten sich aufgrund ihrer freien Verfügbarkeit als Basis für einen Vergleich von Meta-Lernern für Klassifikationsaufgaben an. Von besonderem Interesse wäre ein solcher Vergleich von Meta-Lernern, die die bisher nicht nur in Medam fehlende Parametrisierung der Algorithmen integrieren.

Anhang

A Wetter-Datensatz

Für Beispielberechnungen wird in dieser Arbeit der Wetter-Datensatz verwendet. Damit die Berechnungen nachvollziehbar sind, werden in Tabelle 12 die 14 Instanzen des Datensatzes aufgelistet.

outlook	temperature	humidity	windy	play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

Tabelle 12: Wetter-Datensatz

B Berechnung der Gewinnquote

Die Berechnung der Kennzahl “Durchschnittlich Gewinnquoten-Differenz” aus Abschnitt 4.4.2 erfordert die Berechnung von Gain-Ratio-Werten. Wie man die Gewinnquote (gain-ratio) ermittelt, wird nun dargelegt.

Tabelle 13 zeigt eine Übersicht der im folgenden verwendeten Variablen.

Variable	Erklärung
T	repäsentiert die gegebene Klassifikationsaufgabe (Task).
$m = \|T\|$	ist die Anzahl der Instanzen von T.
a	bezeichnet die Attribute von T.
v_{i_x}	ist der x-te Wert des Attributes a_i, mit $1 \leq x \leq j_i$, wobei j_i die Anzahl der verschiedenen Werte für a_i ist.
$T_{v_{i_x}}$	ist die Menge von Instanzen, für die a_i den x-ten Wert annimmt ($m(v_{i_x})$), wobei $m(v_{i_x}) \leq m$ gilt.
l_k	ist ein Klassenlabel, mit $1 \leq k \leq p$, wobei p die Anzahl der zu lernenden Klassen ist.
T_{l_k}	ist die Menge von Instanzen, die mit der Klasse l_k gekennzeichnet sind, wobei $m(l_k) \leq m$ gilt.

Tabelle 13: Variablen zur Berechnung der Gewinnquote

Folgender allgemeiner Zusammenhang lässt sich für die Anzahl der Instanzen feststellen:

$$\sum_{k=1}^{p} m(l_k) = \sum_{x=1}^{j_i} m(v_{i_x}) = m\,.$$

Um nun herauszufinden, wie informativ jedes Attribut a_i ist, wird das Maß *gain* berechnet:

$$gain(a_i, T) = info(T) - \sum_{x=1}^{j_i} \frac{m(v_{i_x})}{m}\, info(T_{v_{i_x}})\,.$$

Das Informationsmaß *info(S)*, wobei *S* eine gegebene Menge von Instanzen ist, wird wie folgt bestimmt:

$$info(S) = -\sum_{k=1}^{p} \frac{m(l_k)}{m}\, log_2\left(\frac{m(l_k)}{m}\right)\,.$$

m berechnet sich in diesem Fall aus $m = |S|$.
Die Weiterentwicklung des Gain-Maßes führt zum gesuchten Gain-Ratio-Maß:

$$\textit{gain-ratio}(a_i, T) = \frac{gain(a_i, T)}{\textit{split info}(a_i, T)}\,.$$

Dieses Maß korrigiert das Gain-Maß für Attribute, die im Verhältnis zur Anzahl der Trainingsinstanzen eine zu große Anzahl von Werten besitzen. Das zur Berechnung verwendete Split-Info-Maß ergibt sich aus:

$$\textit{split info}(a_i, T) = -\sum_{x=i}^{j_i} \left(\frac{m(v_{i_x})}{m} \, log_2 \left(\frac{m(v_{i_x})}{m} \right) \right) .$$

Ist ein Attribut numerisch, so wird (im Fall vom C4.5-Algorithmus) in Abhängigkeit der Attributverteilung und der Klassenzugehörigkeit der Werte ein geeignetes Split Model gesucht. Kann ein solches Modell nicht gefunden werden, so ist der Gain-Ratio-Wert für dieses Attribut 0.

Beispiel: Zur Verdeutlichung wird der Gain-Ratio-Wert für das Attribut *outlook* des Wetter-Datensatzes (W) berechnet.

$$info(W) = -\left(\frac{9}{14} \, log_2 \left(\frac{9}{14} \right) + \frac{5}{14} \, log_2 \left(\frac{5}{14} \right) \right) \approx 0.9403$$

$$info(W_{v_{outlook_{sunny}}}) = -\left(\frac{2}{5} \, log_2 \left(\frac{2}{5} \right) + \frac{3}{5} \, log_2 \left(\frac{3}{5} \right) \right) \approx 0.971$$

$$info(W_{v_{outlook_{overcast}}}) = -\left(\frac{4}{4} \, log_2 \left(\frac{4}{4} \right) + \frac{0}{4} \, log_2 \left(\frac{0}{4} \right) \right) = 0$$

$$info(W_{v_{outlook_{rainy}}}) = -\left(\frac{3}{5} \, log_2 \left(\frac{3}{5} \right) + \frac{2}{5} \, log_2 \left(\frac{2}{5} \right) \right) \approx 0.971$$

$$gain(a_{outlook}) = 0.9403 - (\frac{5}{14} \, 0.971 + \frac{4}{14} \, 0 + \frac{5}{14} \, 0.971) \approx 0.2467$$

$$\textit{split info}(a_{outlook}, W) = -(\frac{5}{14} \, log_2 \frac{5}{14} + \frac{4}{14} \, log_2 \frac{4}{14} + \frac{5}{14} \, log_2 \frac{5}{14}) \approx 1,5774$$

$$\boxed{\textit{gain-ratio}(a_{outlook}, W) = \frac{0.2467}{1,5774} \approx 0.1564}$$

C Überblick über verwendete Datensätze aus dem UCI-Datenbestand

Datensatz	Instanzanzahl	Attributanzahl	fehlende Werte
allbp	2800	29	√
allhyper	2800	29	√
allhypo	2800	29	√
balance-scale	625	4	-
contact-lenses	24	4	-
dis	2800	29	√
glass	214	10	-
heart	270	13	-
house-votes-84	435	16	√
ionosphere	351	34	-
iris	150	4	-
labor	57	16	-
sick	2800	29	√
soybean	683	35	√
tic-tac-toe	958	9	-
waveform	5000	21	-
weather	14	4	-
wine	178	13	-
zoo	101	16	-

Tabelle 14: Verwendete Datensätze

D Testergebnisse

Hier sind die Ergebnisse in Tabellenform dargestellt, die beim Testen des Meta-Lerners erzielt wurden und in Abschnitt 5.4 ausgewertet wurden.

Datensatz: zoo
Instanzanzahl: 101
Attributanzahl: 16

timeRelevance		10		1		0.1
1	**NB:**	1.66	IB1:	1.67	IB1:	1.69
2	IB1:	1.54	**NB:**	1.65	NB:	1.65
3	J48:	1.41	J48:	1.61	**J48:**	1.63
4	PART:	1.32	PART:	1.6	PART:	1.63
5	OneR:	1.22	OneR:	1.19	OneR:	1.19
6	DS:	1.02	DS:	0.97	DS:	0.97
7	ZeroR:	0.81	ZeroR:	0.6	ZeroR:	0.59
CBR-k=1		J48		J48*		J48*
CBR-k=7		**NB***		**NB***		**J48***
Z_2(ARR)		J48		J48*		J48*
C4.5		IB1		IB1*		IB1*

Tabelle 15: Ergebnistabelle vom Datensatz *zoo*

Datensatz: tic-tac-toe
Instanzanzahl: 958
Attributanzahl: 9

timeRelevance		10		1		0.1
1	ZeroR:	1.41	PART:	1.53	PART:	1.57
2	OneR:	1.33	J48:	1.35	J48:	1.37
3	DS:	1.33	IB1:	1.23	IB1:	1.26
4	**NB:**	1.26	OneR:	1.13	OneR:	1.12
5	PART:	1.25	DS:	1.13	DS:	1.12
6	J48:	1.24	**NB:**	1.11	**NB:**	1.1
7	IB1:	0.99	ZeroR:	1.04	ZeroR:	1.02
CBR-k=1		NB		NB		NB
CBR-k=7		**NB**		**NB**		**NB**
Z_2(ARR)		NB		NB		NB
C4.5		NB		NB		NB

Tabelle 16: Ergebnistabelle vom Datensatz *tic-tac-toe*

Datensatz: weather
Instanzanzahl: 14
Attributanzahl: 4

timeRelevance		10		1		0.1
1	IB1:	2.24	IB1:	2.08	IB1:	2.07
2	J48:	1.3	J48:	1.42	J48:	1.44
3	ZeroR:	1.25	ZeroR:	1.16	ZeroR:	1.15
4	**NB:**	1.18	**NB:**	1.1	**NB:**	1.09
5	DS:	1.1	PART:	1.07	PART:	1.08
6	PART:	1.0	DS:	1.02	OneR:	1.01
7	OneR:	0.93	OneR:	1	DS:	1.01
CBR-k=1		NB		NB		NB
CBR-k=7		**NB**		**NB**		**NB**
Z_2(ARR)		NB		PART		PART
C4.5		NB		NB		NB

Tabelle 17: Ergebnistabelle vom Datensatz *weather*

Datensatz: soybean
Instanzanzahl: 683
Attributanzahl: 35

timeRelevance	10		1		0.1	
1	NB:	2.94	NB:	3.13	NB:	3.17
2	J48:	2.5	**J48:**	3.06	**J48:**	3.14
3	IB1:	2.35	IB1:	3.02	IB1:	3.12
4	PART:	2.23	PART:	3.0	PART:	3.11
5	OneR:	1.29	OneR:	1.23	OneR:	1.23
6	DS:	0.81	DS:	0.79	DS:	0.79
7	**ZeroR:**	0.41	ZeroR:	0.29	ZeroR:	0.28
CBR-k=1		ZeroR		ZeroR		J48*
CBR-k=7		**ZeroR**		**J48***		**J48***
Z_2(ARR)		ZeroR		NB*		NB*
C4.5		NB*		NB*		NB*

Tabelle 18: Ergebnistabelle vom Datensatz *soybean*

Datensatz: sick
Instanzanzahl: 2800
Attributanzahl: 29

timeRelevance	10		1		0.1	
1	**ZeroR:**	2.04	OneR:	1.22	J48:	1.24
2	OneR:	1.43	J48:	1.22	PART:	1.23
3	DS:	1.42	DS:	1.22	OneR:	1.21
4	NB:	1.35	**ZeroR:**	1.21	DS:	1.21
5	J48:	1.13	PART:	1.21	IB1:	1.19
6	PART:	1.08	NB:	1.17	**ZeroR:**	1.17
7	IB1:	0.88	IB1:	1.15	NB:	1.16
CBR-k=1		ZeroR*		ZeroR*		J48*
CBR-k=7		**ZeroR***		**ZeroR***		**ZeroR**
Z_2(ARR)		ZeroR*		ZeroR*		J48*
C4.5		ZeroR*		ZeroR*		J48*

Tabelle 19: Ergebnistabelle vom Datensatz *sick*

Datensatz: iris
Instanzanzahl: 150
Attributanzahl: 4

timeRelevance		10		1		0.1
1	**NB:**	1.7	NB:	1.68	NB:	1.68
2	OneR:	1.57	J48:	1.66	J48:	1.68
3	J48:	1.52	**IB1:**	1.65	PART:	1.67
4	IB1:	1.51	PART:	1.64	**IB1:**	1.66
5	PART:	1.41	OneR:	1.62	OneR:	1.63
6	DS:	1.12	DS:	1.09	DS:	1.09
7	ZeroR:	0.57	ZeroR:	0.46	ZeroR:	0.45
CBR-k=1		NB*		IB1*		IB1*
CBR-k=7		**NB***		**IB1***		**IB1***
Z_2(ARR)		NB*		J48*		PART*
C4.5		NB*		NB*		NB*

Tabelle 20: Ergebnistabelle vom Datensatz *iris*

Datensatz: dis
Instanzanzahl: 2800
Attributanzahl: 29

timeRelevance		10		1		0.1
1	**ZeroR:**	2.14	ZeroR:	1.25	**J48:**	1.22
2	OneR:	1.41	OneR:	1.22	ZeroR:	1.21
3	DS:	1.39	DS:	1.22	OneR:	1.21
4	NB:	1.32	**J48:**	1.21	PART:	1.21
5	J48:	1.16	PART:	1.19	DS:	1.21
6	PART:	1.08	NB:	1.17	IB1:	1.2
7	IB1:	0.87	IB1:	1.16	NB:	1.16
CBR-k=1		ZeroR*		ZeroR*		J48*
CBR-k=7		**ZeroR***		**J48***		**J48***
Z_2(ARR)		ZeroR*		ZeroR*		J48*
C4.5		ZeroR*		J48*		J48*

Tabelle 21: Ergebnistabelle vom Datensatz *dis*

Datensatz: contact-lenses
Instanzanzahl: 24
Attributanzahl: 4

timeRelevance		10		1		0.1
1	J48:	1.57	J48:	1.63	J48:	1.64
2	PART:	1.43	PART:	1.62	PART:	1.64
3	OneR:	1.25	OneR:	1.22	OneR:	1.21
4	DS:	1.25	DS:	1.22	DS:	1.21
5	IB1:	1.18	IB1:	1.14	IB1:	1.14
6	**NB:**	1.11	**NB:**	1.08	**NB:**	1.08
7	ZeroR:	0.86	ZeroR:	0.84	ZeroR:	0.83
CBR-k=1		NB		IB1		IB1
CBR-k=7		**NB**		**NB**		**NB**
Z_2(ARR)		NB		NB		IB1
C4.5		OneR		OneR		OneR

Tabelle 22: Ergebnistabelle vom Datensatz *contact-lenses*

Datensatz: allhypo
Instanzanzahl: 2800
Attributanzahl: 29

timeRelevance		10		1		0.1
1	**ZeroR:**	1.93	**J48:**	1.25	**J48:**	1.26
2	OneR:	1.41	PART:	1.25	PART:	1.26
3	DS:	1.38	OneR:	1.23	OneR:	1.22
4	NB:	1.33	DS:	1.22	DS:	1.21
5	J48:	1.23	NB:	1.2	NB:	1.19
6	PART:	1.2	ZeroR:	1.19	ZeroR:	1.16
7	IB1:	0.81	IB1:	1.08	IB1:	1.12
CBR-k=1		ZeroR*		ZeroR*		J48*
CBR-k=7		**ZeroR***		**J48***		**J48***
Z_2(ARR)		ZeroR*		ZeroR*		J48*
C4.5		ZeroR*		ZeroR*		J48*

Tabelle 23: Ergebnistabelle vom Datensatz *allhypo*

Datensatz: allhyper
Instanzanzahl: 2800
Attributanzahl: 29

timeRelevance	10		1		0.1	
1	**ZeroR:**	2.12	ZeroR:	1.24	OneR:	1.21
2	OneR:	1.43	OneR:	1.22	**J48:**	1.21
3	DS:	1.4	DS:	1.21	PART:	1.21
4	NB:	1.34	**J48:**	1.2	ZeroR:	1.2
5	J48:	1.14	NB:	1.19	DS:	1.2
6	PART:	1.06	PART:	1.19	IB1:	1.19
7	IB1:	0.87	IB1:	1.15	NB:	1.18
CBR-k=1		ZeroR*		ZeroR*		J48*
CBR-k=7		**ZeroR***		**J48***		**J48***
Z_2(ARR)		ZeroR*		ZeroR*		J48*
C4.5		ZeroR*		ZeroR*		J48*

Tabelle 24: Ergebnistabelle vom Datensatz *allhyper*

Datensatz: labor
Instanzanzahl: 57
Attributanzahl: 16

timeRelevance	10		1		0.1	
1	**NB:**	1.61	NB:	1.51	NB:	1.5
2	IB1:	1.36	**IB1:**	1.37	**IB1:**	1.38
3	ZeroR:	1.25	PART:	1.26	PART:	1.29
4	DS:	1.19	J48:	1.22	J48:	1.24
5	OneR:	1.13	DS:	1.14	DS:	1.13
6	J48:	1.09	OneR:	1.06	OneR:	1.05
7	PART:	1.05	ZeroR:	0.98	ZeroR:	0.96
CBR-k=1		J48		J48		J48
CBR-k=7		**NB***		**IB1**		**IB1**
Z_2(ARR)		J48		J48		PART
C4.5		NB*		NB*		NB*

Tabelle 25: Ergebnistabelle vom Datensatz *labor*

Datensatz: heart
Instanzanzahl: 270
Attributanzahl: 13

timeRelevance	10		1		0.1	
1	NB:	1.55	NB:	1.44	NB:	1.43
2	OneR:	1.38	PART:	1.27	PART:	1.3
3	**ZeroR:**	1.34	IB1:	1.25	IB1:	1.27
4	DS:	1.32	OneR:	1.22	**J48:**	1.24
5	IB1:	1.12	**J48:**	1.22	DS:	1.21
6	J48:	1.09	DS:	1.22	OneR:	1.2
7	PART:	1.03	ZeroR:	0.91	ZeroR:	0.88
CBR-k=1		ZeroR		J48		J48
CBR-k=7		**ZeroR**		**J48**		**J48**
Z_2(ARR)		ZeroR		ZeroR		J48
C4.5		ZeroR		PART		PART

Tabelle 26: Ergebnistabelle vom Datensatz *heart*

Datensatz: allbp
Instanzanzahl: 2800
Attributanzahl: 29

timeRelevance	10		1		0.1	
1	**ZeroR:**	2.24	ZeroR:	1.24	J48:	1.22
2	OneR:	1.46	OneR:	1.22	PART:	1.22
3	DS:	1.42	J48:	1.2	OneR:	1.21
4	NB:	1.38	DS:	1.2	IB1:	1.2
5	J48:	1.11	PART:	1.19	ZeroR:	1.2
6	PART:	1.02	**NB:**	1.18	DS:	1.19
7	IB1:	0.89	IB1:	1.16	**NB:**	1.17
CBR-k=1		ZeroR*		J48*		J48*
CBR-k=7		**ZeroR***		**NB***		**NB***
Z_2(ARR)		NB		NB*		J48*
C4.5		NB		NB*		NB*

Tabelle 27: Ergebnistabelle vom Datensatz *allbp*

Datensatz: glass
Instanzanzahl: 214
Attributanzahl: 10

timeRelevance	10		1		0.1	
1	OneR:	1.78	OneR:	1.67	OneR:	1.66
2	**NB:**	1.43	J48:	1.63	PART:	1.66
3	J48:	1.43	PART:	1.62	**J48:**	1.65
4	IB1:	1.37	**IB1:**	1.54	IB1:	1.56
5	PART:	1.35	NB:	1.44	NB:	1.45
6	DS:	1.04	DS:	1.09	DS:	1.1
7	ZeroR:	0.76	ZeroR:	0.51	ZeroR:	0.49
CBR-k=1		NB		NB		NB
CBR-k=7		**NB**		**IB1**		**J48***
Z_2(ARR)		NB		NB		IB1
C4.5		J48		J48*		J48*

Tabelle 28: Ergebnistabelle vom Datensatz *glass*

Datensatz: ionosphere
Instanzanzahl: 351
Attributanzahl: 34

timeRelevance	10		1		0.1	
1	**ZeroR:**	1.62	PART:	1.32	PART:	1.35
2	OneR:	1.38	J48:	1.31	J48:	1.34
3	NB:	1.29	**IB1:**	1.26	**IB1:**	1.28
4	DS:	1.25	OneR:	1.24	OneR:	1.24
5	IB1:	1.13	DS:	1.22	DS:	1.22
6	J48:	1.13	NB:	1.21	NB:	1.21
7	PART:	1.09	ZeroR:	0.92	ZeroR:	0.89
CBR-k=1		OneR		OneR		J48*
CBR-k=7		**ZeroR***		**IB1***		**IB1***
Z_2(ARR)		NB		J48*		J48*
C4.5		NB		NB		NB

Tabelle 29: Ergebnistabelle vom Datensatz *ionosphere*

Datensatz: waveform
Instanzanzahl: 5000
Attributanzahl: 21

timeRelevance		10		1		0.1
1	**NB:**	1.88	**NB:**	1.69	**NB:**	1.69
2	ZeroR:	1.38	PART:	1.57	PART:	1.63
3	J48:	1.35	IB1:	1.55	IB1:	1.6
4	DS:	1.25	J48:	1.55	J48:	1.58
5	OneR:	1.22	DS:	1.13	DS:	1.13
6	PART:	1.18	OneR:	1.06	OneR:	1.06
7	IB1:	1.19	ZeroR:	0.62	ZeroR:	0.59
CBR-k=1		NB*		NB*		NB*
CBR-k=7		**NB***		**NB***		**NB***
Z_2(ARR)		NB*		PART		PART*
C4.5		PART		PART		PART*

Tabelle 30: Ergebnistabelle vom Datensatz *waveform*

Datensatz: wine
Instanzanzahl: 178
Attributanzahl: 13

timeRelevance		10		1		0.1
1	**NB:**	1.66	NB:	1.67	NB:	1.68
2	IB1:	1.44	PART:	1.63	PART:	1.66
3	J48:	1.41	J48:	1.62	J48:	1.65
4	PART:	1.39	**IB1:**	1.61	**IB1:**	1.64
5	OneR:	1.38	OneR:	1.32	OneR:	1.32
6	DS:	0.93	DS:	0.96	DS:	0.97
7	ZeroR:	0.85	ZeroR:	0.57	ZeroR:	0.55
CBR-k=1		NB*		NB*		NB*
CBR-k=7		**NB***		**IB1***		**IB1***
Z_2(ARR)		J48		J48*		J48*
C4.5		ZeroR		ZeroR		J48*

Tabelle 31: Ergebnistabelle vom Datensatz *wine*

Datensatz: balance-scale
Instanzanzahl: 625
Attributanzahl: 4

timeRelevance	10		1		0.1	
1	**NB:**	2.02	**NB:**	1.83	**NB:**	1.82
2	OneR:	1.3	PART:	1.48	PART:	1.52
3	DS:	1.21	IB1:	1.26	IB1:	1.28
4	PART:	1.18	J48:	1.22	J48:	1.23
5	J48:	1.13	OneR:	1.12	OneR:	1.11
6	ZeroR:	1.1	DS:	1.05	DS:	1.04
7	IB1:	1.07	ZeroR:	0.84	ZeroR:	0.82
CBR-k=1		ZeroR		PART		PART
CBR-k=7		**NB***		**NB***		**NB***
Z_2(ARR)		NB*		PART		PART
C4.5		PART		PART		PART

Tabelle 32: Ergebnistabelle vom Datensatz *balance-scale*

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[ADL98] Arne Andersson, Paul Davidsson, and Johan Lindén. Model selection using measure functions. In *Proc. of the ECML'98 Workshop on Upgrading Learning to the Meta-Level: Model Selection and Data Transformation*, 1998.

[BBG+99] R. Bergmann, S. Breen, M. Göker, M. Manago, and S. Wess. The INRECA-Methodology. In *Developing Industrial Case-Based Reasoning Applications*, volume 1612 of *Lecture Notes in Artificial Intelligence*. Springer, 1999.

[Ben98] Hilan Bensusan. God Doesn't Always Shave with Occam's Razor - Learning When and How to Prune. In *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Artificial Intelligence*, pages 119–124, Chemnitz, 1998. Springer.

[Ben99] Hilan N. Bensusan. *Automatic bias learning: An inquiry into the inductive basis of induction*. PhD thesis, University of Sussex, 1999.

[BGC00a] Hilan Bensusan and Christophe Giraud-Carrier. Casa Batló is in Passeig de Gràcia or how landmark performances can describe tasks. ECML'2000, 2000. http://citeseer.nj.nec.com/bensusan00casa.html.

[BGC00b] Hilan Bensusan and Christophe Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 325–330, Lyon, 2000. Springer.

[BGC00c] Hilan Bensusan and Christophe Giraud-Carrier. Harmonia loosely praestabilita: discovering adequate inductive strategies. In *Proc. of the 22nd Annual Meeting of the Cognitive Science Society*, 2000.

[BGCK00] Hilan Bensusan, Christophe Giraud-Carrier, and C.J. Kennedy. A Higher-order Approach to Meta-learning. In *Proc. of the ECML'2000 workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*, June 2000.

[BGCP+00] Hilan Bensusan, Christophe Giraud-Carrier, Bernhard Pfahringer, Carlos Soares, and Pavel Brazdil. What works well tells us what works better. In *Proc. of ICML'2000 workshop on What Works Well Where*, June 2000.

[BP] Stephen D. Bay and Michael J. Pazzani. Characterizing Model Performance in the Feature Space. http://citeseer.nj.nec.com/355179.html.

[BPK] H. Berrer, Iain Paterson, and Jörg Keller. Evaluation of Machine-Learning Algorithm Ranking Advisors. http://citeseer.nj.nec.com/435201.html.

[BS00] Pavel B. Brazdil and Carlos Soares. A Comparison of Ranking Methods for Classification Algorithm Selection. In *Machine Learning: ECML 2000*, volume 1810 of *Lecture Notes in Artificial Intelligence*, pages 63–74, Barcelona, 2000. Springer.

[C50] ComparisionC4.5/C5.0. www.rulequest.com/see5-comparison.html.

[CAR] CART. www.salford-systems.com.

[Car90] J.G. Carbonell, editor. *Machine Learning Paradigms and Methods*. The MIT Press, 1990.

[Cha98] Don Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann, 1998.

[CS97] Philip K. Chan and Salvatore J. Stolfo. On the Accuracy of Meta-learning for Scalable Data Mining. *Journal of Intelligent Information Systems*, 8(1), 1997. http://citeseer.nj.nec.com/chan96accuracy.html.

[FI92] U.M. Fayyad and K.B. Irani. The Attribute Selection Problem in Decision Tree Generation. In *Proceedings of the Tenth National Conference on Artificial Intelligence AAAI-92*, pages 104–110. AAAI Press, 1992.

[FPSS96] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, Portland, August 1996. AAAI Press.

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

[FSC] Wei Fan, Salvatore J. Stolfo, and Philip K. Chan. Using Conflicts Among Multiple Base Classifiers to Measure the Performance of Stacking. http://citeseer.nj.nec.com/298048.html.

[GB95] J. Gama and P. Brazdil. Characterization of Classification Algorithms. In C. Pinto-Ferreira and N.J. Mamede, editors, *Progress in Artificial Intelligence*, 1995.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, June 1998.

[Her97] Jürgen Herrmann. *Maschinelles Lernen und Wissensbasierte Systeme*. Springer-Verlag, 1997.

[HGN00] J. Hipp, U. Gützer, and G. Nakhaeizadeh. Mining Assoziation Rules: Deriving a Superior Algorithm by Analyzing Today's Approaches. In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 159–168. Springer, 2000.

[HK00a] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[HK00b] Melanie Hilario and Alexandros Kalousis. Building Algorithm Profiles for Prior Model Selection in Knowledge Discovery Systems, 2000.

[HK01] Melanie Hilario and Alexandros Kalousis. Fusion of Meta-Knowledge and Meta-Data for Case-Based Model Selection. Technical report, University of Geneva, 2001. UNIGE-AI-01-01.

[HS91] Marcel Holsheimer and Arno Siebes. Data Mining - The Search for Knowledge in Databases, 1991.

[IBM99] IBM. *Using the Intelligent Miner for Data*, ibm db2 intelligent miner for data edition, 1999.

[KH00a] A. Kalousis and M. Hilario. Model Selection via Meta-learning: a Comparative Study. In *Proceedings of the 12th International IEEE Conference on Tools with AI*, 2000.

[KH00b] A. Kalousis and M. Hilario. Model Selection via Meta-learning: a Comparative Study, 2000.

[KH01] Alexandros Kalousis and Melanie Hilario. Feature Selection for Meta-learning. In *Proc. of the 5th PAKDD*, Hong Kong, April 2001.

[KMR+94] Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, Hannu Toivonen, and A. Inkeri Verkamo. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *3rd Int. Conf. on Information and Knowledge Management*. ACM Press, Nov-Dec 1994.

[KT99] A. Kalousis and T. Theoharis. Noemon: Design, Implementation and Performance Results of an Intelligent Assistant for Classifier Selection. *Intelligent Data Analysis*, 1999.

[LS99] Guido Lindner and Rudi Studer. AST: Support for Algorithm Selection with a CBR Approach. In *Principles of Data Mining and Knowledge Discovery*, volume 1704 of *Lecture Notes in Artificial Intelligence*, pages 418–423, Prague, 1999. Springer.

[MA94] P.M. Murphy and D.W. Aha. UCI Repository of machine learning databases. http://www.ics.uci.edu/˜ mlearn/MLRepository.html, 1994. Irvine, CA: University of California, Department of Information and Computer Science.

[MBK01] R.S. Michalski, I. Bratko, and M. Kubat. *Machine Learning and Data Mining*. John Wiley & Sons Ltd, 2001.

[Met01] ESPRIT METAL Project (26.357), Dec.1998-Nov.2001. Project Homepage. http://www.metal-kdd.org/.

[Mit97] Tom M. Mitchell. *Machine Learning*. The WCB/McGraw-Hill Companies, 1997.

[MN88] P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection*. North-Holland, Amsterdam, 1988.

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

[Mue95] Andreas Mueller. Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison, 1995.

[Noc00] Thomas Nocke. Metadatengewinnung und -spezifikation für Visualisierungsentscheidungen. Master's thesis, Universität Rostock, 2000.

[PBGC00] Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier. Tell me who can learn you and I can tell you who you are: Landmarking Various Learning Algorithms. In *Machine Learning: Proc. of the Int. Conf. on Machine Learning ICML'2000*, 2000.

[Rei99] Thomas Reinartz. *Focusing Solutions for Data Mining*, volume 1623 of *Lecture Notes in Artificial Intelligence*. Springer, 1999.

[RF00] C. Robardet and F. Feschet. A New Methodology to Compare Clustering Algorithms. In *Intelligent Data Engineering and Automated Learning - IDEAL 2000*, volume 1983 of *Lecture Notes in Computer Science*, pages 565–570. Springer, 2000.

[Ric95] Michael M. Richter. The Knowledge Contained in Similarity Measures, 1995. Invitited talk given at ICCBR'95 in Sesimbra, Portugal.

[SB00] Carlos Soares and Pavel B. Brazdil. Zoomed Ranking: Selection of Classification Algorithms Based on Relevant Performance Information. In *Principles of Data Mining and Knowledge Discovery*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 126–135, Lyon, 2000. Springer.

[SCB] Carlos Soares, J. Costa, and Pavel B. Brazdil. A Simple and Intuitive Measure for Multicriteria Evaluation of Classification Algorithms. http://citeseer.nj.nec.com/391351.html.

[SH99] Gunter Saake and Andreas Heuer. *Datenbanken: Implementierungstechniken*. MITP, 1999.

[SUD98] A. Seitz, A.M. Uhrmacher, and D. Damm. Case-Based Prediction in Experimental Medical Studies. *The International Journal on AI in Medicine*, 1998. http://citeseer.nj.nec.com/article/seitz98casebased.html.

[Syk] Peter Sykacek. Metalevel learning - is more than model selection necessary? http://citeseer.nj.nec.com/311118.html.

[WF01] Ian H. Witten and Eibe Frank. *Data Mining - Praktische Werkzeuge und Techniken für das maschinelle Lernen*. Carl Hanser Verlag, 2001.

[WM95] D. Wolpert and W. Macready. No Free Lunch Theorem for Search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[WSG+97] Rüdiger Wirth, Colin Shearer, Udo Grimmer, Thomas Reinartz, Jörg Schlösser, Christoph Breitner, Robert Engels, and Guido Lindner. Towards Process-Oriented Tool Support for Knowledge Discovery in Databases. In *Principles of Data Mining and Knowledge Discovery*. 1997. http://citeseer.nj.nec.com/112592.html.

[Xia] Zhihua Xiao. Association Rules Mining Algorithm. Department of Information System and Computer Science, National University of Singapore.

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, den 6. November 2001

Thesen

1. Unterschiedliche Problemstellungen des Data Mining erfordern unterschiedliche Verfahren zur Problemlösung.

2. Die Vielfalt und Komplexität von Data-Mining-Verfahren lässt ein intelligentes Softwaresystem zur Auswahlunterstützung als wichtig erscheinen. Dazu sind prinzipiell Verfahren wie Expertensysteme und Lernsysteme geeignet.

3. Der Vorteil der Lernsysteme als Auswahlunterstützung liegt darin, dass sie durch Beobachtung der Data-Mining-Verfahren automatisch lernen, in welchen Situationen welche Algorithmen anzuwenden sind.

4. Die meisten Problemstellungen des Data Mining lassen sich den Aufgabentypen Klassifikation, Assoziation oder Clustering zuordnen.

5. Während sich eine Meta-Lerner-Architektur allgemein für alle drei Aufgabentypen angeben lässt, erfordert die Anwendbarkeit des Meta-Lerners eine Aufgaben-spezifische Instanziierung der eingebundenen Komponenten.

6. Verglichen mit anderen Verfahrensfamilien finden sich in der Klassifikation überdurchschnittlich viele Algorithmen, daher bietet es sich an, den Meta-Lerner für die Klassifikation prototypisch zu entwickeln.

7. Der Erfolg des Meta-Lerners hängt ab von dem eingesetzten Lernverfahren, von der Beschreibung der Problemstellung und den zur Auswahl stehenden Verfahren.

8. Die Beschreibung von Klassifikationsaufgaben auf der Basis von Entscheidungsbäumen bietet Vorteile, da sie implizite Eigenschaften mit Hilfe eines Klassifikationsverfahren hervorbringt und für Attribute unterschiedlichen Skalenniveaus gleichermaßen anwendbar ist.

9. Fallbasierte Verfahren bieten sich an, wenn aus einer kleineren Datenmenge die beste Lösung nicht nur durch Selektion, sondern auch durch Adaptation ermittelt werden soll. Als Lernverfahren für Meta-Lerner sind sie besonders gut geeignet, da sie der Selektion von Data-Mining-Verfahren und deren Parameter dienen.

10. Das Problem fallbasierter Verfahren liegt in der Gleichgewichtung von Attributen, was unter Umständen die Performanz des Verfahrens beeinträchtigt. Dem kann durch das informationstheoretische Bewerten der Attribute begegnet werden.

11. Die in Medam prototypisch umgesetzte Architektur unter Integration des fallbasierten Ansatzes und der Entscheidungsbaum-basierten Aufgabenbeschreibung ist erfolgversprechend und kann als Vorlage für weitere Meta-Lerner dienen.