Eine Untersuchung des Trade-Offs zwischen Precision und Coverage bei Regel-Lern-Heuristiken

Diplomarbeit

im Studiengang `Informatik`

angefertigt am Fachgebiet Knowledge Engineering

der Technischen Universität Darmstadt

von

`Frederik Janssen`

`Juni 2006`

Betreuer: Prof. J. Fürnkranz

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 16. Juni 2006 Frederik Janssen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Ein noch immer ungelöstes Problem im Bereich des induktiven Regel-Lernens ist das Finden einer optimalen Heuristik. Da diese die Güte der gefundenen Regeln bewertet, ist sie ein wichtiger Teil des gesamten Regel-Lernens. Üblicherweise hat eine Heuristik, die eher allgemeine Regeln findet, den Vorteil, dass die gefundenen Regeln viele Beispiele abdecken und den Nachteil, dass die Regeln nicht sehr genau sind. Andererseits hat eine Heuristik, die eher genaue Regeln findet, den Nachteil, dass durch diese Regeln nicht so viele Beispiele abgedeckt werden. Daraus entsteht ein Trade-Off, der angibt, wie genau und wie allgemein eine Regel ist. Optimal wäre eine Regel, die möglichst alle Beispiele abdeckt aber dabei noch sehr genau ist.

In dieser Diplomarbeit werden Heuristiken untersucht, mit denen mittels eines Parameters ein Trade-Off zwischen Genauigkeit und Abdeckung justiert werden kann. Bei diesen parametrisierbaren Heuristiken werden zur Evaluation der Abdeckung verschiedene Basis-Heuristiken verwendet. Wählt man beispielsweise beim Klösgen-Maß einen sehr geringen Parameter, dann ergibt sich eine Bewertungsfunktion, die eher auf genaue Regeln abzielt. Benutzt man hingegen einen sehr hohen Wert, so werden die gefundenen Regeln allgemeiner, aber auch ungenauer.

1.2 Ziel der Arbeit

In dieser Arbeit sollen in einer empirischen Studie für die drei parametrisierbaren Heuristiken *Klösgen-Maß*, *m-Estimate* und *F-Measure* auf verschiedenen Datenmengen optimale Parameter gefunden und auf disjunkten Datenmengen getestet werden. Es ist bewusst eine

sehr große Anzahl von 27 Datenmengen gewählt worden, um aussagekräftige Ergebnisse zu erhalten. Die gefundenen Parameter sollten eine höhere Genauigkeit als die fünf Standard-Heuristiken aufweisen können und auch in der Lage sein, mit einer Implementierung von *RIPPER* mitzuhalten, der zur Zeit einer der besten Regel-Lerner ist. Dazu wurden alle Heuristiken in einer umfangreichen Studie miteinander verglichen. Dieser Vergleich ist mit unterschiedlichen Evaluationsmethoden durchgeführt worden, um verschiedene Gesichtspunkte zu beleuchten.

1.3 Überblick über die Kapitel

Im 2. Kapitel wird ein Überblick über Regel-Lern-Algorithmen und deren Funktionsweise gegeben. Zu Anfang wird das Ziel eines Lernalgorithmus näher erläutert. Dann folgt eine Einteilung der verschiedenen Separate-and-Conquer Algorithmen und deren Strategie wird aufgezeigt. ROC-Räume zur Evaluation von Klassifikatoren und Maßen werden dargestellt und es wird auf die spezielle Rolle der Heuristiken beim Regel-Lernen eingegangen. Abschließend wird mit einer in Java implementierten Variante von *RIPPER* ein Regel-Lern Algorithmus exemplarisch vorgestellt.

Kapitel drei gibt einen Überblick über die verwendeten Heuristiken. Es folgt die Veranschaulichung einer Heuristik mit Isometriken und die Standard-, Basis- und parametrisierbaren Heuristiken werden ausführlich erläutert.

In Kapitel vier wird eine Definition des experimentellen Szenarios gegeben. Dazu wird kurz auf die verwendeten Datenmengen eingegangen. Außerdem wird die Cross-Validation und der für die Experimente benutzte Regel-Lerner vorgestellt. Zusätzlich wird noch auf die Probleme, die beim Bewerten von Klassifikatoren entstehen, verwiesen. Die Eigenarten, die bei einer Evaluation mit Cross-Validation auftreten, werden ebenfalls skizziert. Am Ende des Kapitels sind die Evaluationsmethoden aufgezeigt, mit denen die Ergebnisse bewertet wurden.

Das 5. Kapitel behandelt ausführlich die erreichten Resultate. Das Verfahren zur Bestimmung der optimalen Parameter wird erklärt, und die Eigenarten und Probleme der parametrisierbaren Heuristiken werden erläutert. Es folgt ein Vergleich der drei Heuristiken untereinander. Des weiteren werden sie mit anderen Standard-Heuristiken verglichen, um die Güte der gefundenen Parametrisierungen besser einschätzen zu können. Auf die Qualität wird dann nochmals bei der Überprüfung der Allgemeingültigkeit unter Verwendung einer unabhängigen Datenmenge besonderes Augenmerk gelegt.

In Kapitel sechs werden alle Ergebnisse abschließend zusammengefasst und ein Ausblick auf weitere interessante Ansätze für die Auswahl der Suchheuristik im Regel-Lernen gegeben.

Kapitel 2

Regel-Lernen

Das Ziel beim Regel-Lernen ist - wie bei allen induktiven Konzept-Lernalgorithmen - eine Theorie zu finden, die die Beispiele einer Trainingsmenge möglichst gut den zugehörigen Klassen zuordnet. Unter einer Theorie versteht man bei Regel-Lernern eine Menge von Regeln. Der Begriff Konzept wird im Folgenden als Synonym für den Begriff Theorie verwendet. Ein Lernalgorithmus ist beim überwachten Lernen (engl. supervised learning) ein Algorithmus, der als Eingabe eine Trainingsmenge von klassifizierten (mit einem Klassenlabel versehenen) Beispielen erhält. Er lernt auf dieser Menge eine Theorie und gibt einen Klassifikator aus. Ein Klassifikator bekommt nun als Eingabe eine Testmenge, die aus Beispielen besteht, die keiner Klasse zugeordnet sind. Für ein gegebenes Beispiel gibt der Klassifikator die Klasse aus, die konsistent mit der Theorie für dieses Beispiel ist. Die einzelnen Beispiele wiederum bestehen aus Attributen, die unterschiedliche Werte annehmen können. Das letzte Attribut entspricht einer Klassenzuordnung. Eine Instanz, also eins der Trainingsbeispiele, kann beispielsweise durch folgenden Vektor repräsentiert werden:

$$(Wetter = sonnig, Temperatur = 28, Wind = ja, SpieleTennis = ja) \tag{2.1}$$

Hier gibt es die Attribute „Wetter“, „Temperatur“ und „Wind“. Unterschieden wird zwischen nominalen und numerischen Attributen. Sobald es eine Relation gibt, die die Werte der Attribute sinnvoll ordnet, handelt es sich um numerische oder auch lineare Attribute[1]. Beim „Wetter“ handelt es sich um ein nominales Attribut welches die Werte „sonnig“, „bewölkt“ oder „regnerisch“ annehmen kann. Die „Temperatur“ ist ein numerisches Attribut, welches zum Beispiel Werte der Menge $\{0, ..., 40\}$ annehmen kann. Als Klassenwert dient die Frage,

[1] Es gibt bei nominalen Attributen keine sinnvolle Ordnungsrelation für die unterschiedlichen Werte. Man könnte beispielsweise Werte des Attributs „Wetter“ nach der „Güte“ des Wetters ordnen (z.B. regnerisch, bewölkt, sonnig - wobei dies einer individuellen Ordnung entspricht, da es Menschen gibt, die z.B. den Regen der Sonne bevorzugen) oder auch eine alphabetische Ordnung versuchen. Doch mathematische Operationen auf nominal skalierten Merkmalen sind nicht sinnvoll durchführbar ohne eine Abbildung auf eine Ordinalskala (deren Merkmale ebenfalls unterschieden werden können, aber sich zusätzlich dazu in eine objektive (allgemeingültige) Rangfolge bringen lassen - und damit nicht mehr subjektiven Präferenzen unterliegen).

ob man bei diesen äußeren Umständen in der Lage ist, Tennis zu spielen, wobei es sich um eine binäre Klasse handelt, da die Antwort auf die Frage nur aus „ja“ oder „nein“ bestehen kann. Bei der weiteren Beschreibung des induktiven Regel-Lernens wird davon ausgegangen, dass es sich immer um 2-Klassen-Probleme handelt, da viele Separate-and-Conquer Algorithmen auf dieser Annahme beruhen. Es wird im Abschnitt 2.2.3 erläutert, dass diese Algorithmen selbstverständlich auch in der Lage sind Mehrklassenprobleme zu lösen.

Eine Theorie sollte durch geschickte Auswertung der Attribute möglichst jedem Beispiel den richtigen Klassenwert zuordnen. Möchte man nun ein Beispiel bewerten, welches noch keinen Klassenwert besitzt, wird der Klassifikator versuchen die gespeicherten Regeln anzuwenden. Sobald eine Regel auf dieses Beispiel zutrifft (in diesem Zusammenhang „feuert“) wird es der entsprechenden Klasse zugeordnet. Eine solche Regel besteht aus einem Regelkörper und einem Regelkopf. Im Regelkörper stehen konjugierte Bedingungen, die verschiedene Attribute testen. Ein Test besteht aus der Abfrage, ob eine Relation erfüllt ist. Hierbei handelt es sich bei nominalen Attributen um die Relationen $=$ und $\neq$ und bei numerischen meistens um $<$ und $>$ oder $\leq$ und $\geq$. Im Regelkopf steht eine Klassenzuweisung. Die Regel, die das Trainingsbeispiel aus (2.1) abdeckt, könnte wie folgt aussehen:

$$(Wetter = sonnig \wedge Temperatur > 27 \wedge Wind = ja) \rightarrow SpieleTennis = ja. \qquad (2.2)$$

Aufgrund der Einordnung von so genannten Testbeispielen, von denen der korrekte Klassenwert bekannt ist, dem Klassifikator aber vorenthalten wird, kann die Genauigkeit einer Theorie berechnet werden. Diese ergibt sich aus dem Anteil von richtig klassifizierten Beispielen im Verhältnis zu allen Beispielen:

Definition 2.1 (Genauigkeit) $Genauigkeit\,der\,Theorie = \frac{korrekt\,klassifizierte\,Beispiele}{alle\,Beispiele}$

Je nachdem welche Repräsentation der Theorie als Basis dient, ergeben sich verschiedene Vor- und Nachteile. Vergleicht man einen Klassifikator, der auf Entscheidungsbäumen beruht, mit einem, der Regeln verwendet, so fällt auf, dass die Bäume häufig komplexer und für den Benutzer nicht intuitiv verständlich sind. Es wurde in [24] gezeigt, dass ein Entscheidungsbaum der Tiefe k, weniger erklärungsstark ist, als eine Entscheidungsliste (eine geordnete Menge von Regeln), bei der jede Regel höchstens k Bedingungen enthalten darf. Des weiteren ist es bei Entscheidungsbäumen nicht möglich, dass sich Äste überlappen, da jeder Pfad durch den Baum eindeutig ist. Bei Regeln kann eine bestimmte Bedingung in mehreren Varianten vorkommen, weshalb hier eine Überlappung zulässig ist. Regeln sind häufig sehr einfach und intuitiv verständlich. Ein Benutzer ist in der Lage aus den Regeln der Theorie schnell zu erkennen, welche Präferenzen der Klassifikator hat und warum er diese Regeln gefunden hat, um die Trainingsmenge zu beschreiben. Die Regeln sind direkt aus den Daten generierbar und oftmals modular aufgebaut.

Im folgenden Kapitel soll das Lernen von Regeln mit Hilfe der Separate-and-Conquer Strategie erläutert werden. Dazu werden zu Anfang die verschiedenen Methoden zur Einteilungen von Separate-and-Conquer Algorithmen aufgezeigt. Es folgt eine Erklärung des Prinzips, auf dem sie beruhen. Auf die spezielle Rolle einer Heuristik beim Regel-Lernen wird ausführlich eingegangen. Zur Evaluation von Klassifikatoren werden die ROC-Räume erläutert. Außerdem wird als Vertreter eines Algorithmus, der auf dem Separate-and-Conquer Prinzip beruht, dieses aber noch mit Incremental Reduced Error Pruning [12] kombiniert, eine in Java implementierte Variante von *RIPPER* [3] namens JRip [33] näher erläutert.

2.1 Eine Einteilung der Separate-and-Conquer Algorithmen

Es gibt weit über 50 verschiedene Separate-and-Conquer Algorithmen, die sich durch gewisse Merkmale unterscheiden. Eine Auflistung findet sich in [7]. Die wichtigsten Unterscheidungsmöglichkeiten gliedern sich in

1. die verwendete Repräsentationssprache,
2. den Suchalgorithmus, die Suchstrategie und die Suchheuristik und
3. den Mechanismus, der *Overfitting* (Überbestimmtheit) vermeiden soll.

2.1.1 Die verschiedenen Repräsentationssprachen

Die unterschiedlichen Hypothesensprachen gliedern sich in die verschiedenen Konzepte eine Regel zu repräsentieren. Diese können beispielsweise durch *disjunktive Normalform (DNF)*, *konjunktive Normalform (KNF)*, Entscheidungslisten [24], Logik-Programme *(PROLOG)*, funktionale Relationen oder Regressionsregeln beschrieben werden. Bei einer *DNF* handelt es sich um eine Disjunktion von Konjunktionen für jede Klasse. Zusammengesetzt wird sie aus Literalen $L_{i,j}$ und hat folgende Form:

$$(L_{1,1} \wedge L_{1,2} \wedge ... \wedge L_{1,j}) \vee ... \vee (L_{i,1} \wedge L_{i,2} \wedge ... \wedge L_{i,j}) \tag{2.3}$$

Als Literale werden Attributtests verwendet, also beispielsweise *Wetter* = *sonnig*. Die Algorithmen der AQ-Familie [19] benutzen zur Repräsentation der Regeln *DNF*.

Die *KNF* ist eine Negation der *DNF* und besteht aus diesem Grund aus einer Konjunktion von Disjunktionen. Negiert man das obige Beispiel zur *DNF* doppelt, so erhält man denselben Ausdruck in *KNF* :

$$\neg\neg((L_{1,1} \wedge L_{1,2} \wedge ... \wedge L_{1,j}) \vee ... \vee (L_{i,1} \wedge L_{i,2} \wedge ... \wedge L_{i,j})) =$$

$$\neg((\neg L_{1,1} \vee \neg L_{1,2} \vee ... \vee \neg L_{1,j}) \wedge ... \wedge (\neg L_{i,1} \vee \neg L_{i,2} \vee ... \vee \neg L_{i,j})) \tag{2.4}$$

Eine Entscheidungsliste ist eine sortierte Menge von Regeln, die der Reihe nach getestet werden. Sobald eine Regel auf das Beispiel zutrifft, wird der zugehörige Klassenwert ausgegeben und die restlichen Regeln der Liste ignoriert. Als letztes wird eine Default-Regel eingefügt, die keine Bedingungen enthält. Sie trifft immer dann zu, wenn keine der vorherigen Regeln bei dem Beispiel gefeuert hat. Es wird als Klasse üblicherweise die am häufigsten vorhandene zurückgegeben. Es ist möglich Entscheidungslisten in *DNF* zu überführen, allerdings kann man nicht einfach alle Regeln mit $\vee$ verknüpfen, da die Regeln einer solchen Liste nicht allgemeingültig sind. Trifft eine Regel auf ein Beispiel zu, so haben alle vorherigen Regeln nicht zugetroffen, was implizit in der Liste kodiert ist. Daher ist es beispielsweise möglich, dass die 10. Regel der Liste nur noch ein Attribut testet und die anderen außer Acht lässt, da die vorherigen Regeln gezeigt haben, dass die Attribute nicht zutreffen. Aus diesem Grund sind die Voraussetzungen für die aktuelle Regel nur innerhalb der Liste und nicht in der Regel selbst kodiert. Daher sind Entscheidungslisten üblicherweise deutlich kleiner als die Repräsentation in *DNF*. Der bekannteste Algorithmus, der Entscheidungslisten als Repräsentation der Regeln benutzt, ist *CN2* [2].

Logik-Programme in der Sprache *PROGOL* werden in Foil [23] benutzt und funktionale Relationen finden in [22] Verwendung.

Eine weitere Unterscheidung der verschiedenen Konzepte zur Repräsentation der Regeln liegt darin, ob sie statisch oder dynamisch sind. Das Problem einer statischen Methode ist ein Anwachsen des Raumes, in dem sich die Regeln befinden. Wird der Raum durch alle Attribute mit allen möglichen Werten aufgespannt, so wird er schnell zu groß, um noch effizient verwendet werden zu können. Hierfür bieten sich syntaktische Einschränkungen an, die beispielsweise die Anzahl von Bedingungen einer Regel auf einen bestimmten Wert festlegen. Andere Optimierungen werden benötigt um einer zu kurzsichtigen Suche vorzubeugen. So kann es zum Beispiel sinnvoll sein, nicht immer nur eine Bedingung zu einer Regel hinzuzufügen, sondern während eines Schritts auch gleich mehrere konjugierte Bedingungen als Möglichkeit zuzulassen. Verfolgt man diese Richtung weiter, so ist es mit so genannten Regelmodellen möglich, die Regeln anhand einer kontextfreien Grammatik zu finden, die sukzessive alle möglichen Regeln erstellt.

Verwendet man eine dynamische Methode, so kann man beispielsweise alle möglichen Hypothesensprachen in einer hierarchischen Liste absteigend nach ihrer Aussagefähigkeit speichern. Man beginnt mit der obersten, am wenigsten aussagekräftigen Sprache und versucht damit die Trainingsdaten zu erklären. Sobald man merkt, dass dieses Problem nicht zufriedenstellend gelöst werden kann, nimmt man die nächste Sprache in der Liste. Außerdem

ist es möglich die Hypothesensprache automatisch mit neuen Features[2] zu ergänzen, indem diese aus einer Menge von arithmetischen und logischen Operatoren berechnet werden.

2.1.2 Die unterschiedlichen Suchoptionen

Beim Separate-and-Conquer Regel-Lernen wird meistens eines von vier verschiedenen Suchverfahren verwendet. Bei der *Hill-Climbing Suche* wird eine Regel durch das Hinzufügen von Bedingungen sukzessive so lange weiter verfeinert, bis ein globales Optimum gefunden ist. Aus einer Reihe von lokalen Optimierungen entsteht somit eine global beste Theorie. Ein Vorteil dieser Art der Suche ist ihre Effizienz. Ein Nachteil ist jedoch ihr kurzsichtiges Verhalten. Wenn zum Beispiel eine lokal nicht optimale Regel durch weitere Verfeinerungen ein globales Optimum ergäbe, so würde dieses mit der *Hill-Climbing Suche* nicht gefunden werden. Daher liegt die Überlegung nahe, nicht eine einzige Regel pro Schritt zu verfeinern, sondern sich eine Liste mit möglichen Kandidatenregeln zu speichern, damit man nicht so eingeschränkt bleibt.

Dieses Verfahren wird als *Beam-Suche* bezeichnet, wobei der Beam der Liste der Kandidaten entspricht. Ansonsten funktioniert diese Suche analog zur *Hill-Climbing* Variante, weshalb die beiden Suchmethoden für einen Beam mit $b = 1$ gleich sind. Interessant ist, dass die *Beam-Suche* annähernd gleich effizient funktioniert wie die *Hill-Climbing* Methode. Wählt man aber den Beam sehr groß, so ist die *Hill-Climbing Suche* effizienter. Da aber nachgewiesenermaßen ab einer bestimmten Beamgröße sogar schlechtere Ergebnisse erzielt werden [23], kann man davon ausgehen, dass beide Methoden in der Praxis gleich effizient funktionieren, da man den Beam nie so groß wählen wird, dass die Performanz eingeschränkt werden würde. Das Problem der Kurzsichtigkeit bei dieser Art von Suche bleibt prinzipiell allerdings auch bei einem Beam bestehen, lässt sich aber mit diesem Verfahren einschränken.

Wählt man nun den Beam $b = \infty$, so erhält man das Verfahren *Best-First Suche*. Bei dieser Methode werden alle möglichen Kandidatenregeln gespeichert, was selbstverständlich zu dem Problem führt, dass ein riesiger Suchraum entsteht. Durch Anwendung des *A*-Algorithmus* [14] ist es allerdings möglich, durch geschicktes Pruning (siehe 2.1.3) den Suchraum erheblich zu verkleinern, ohne dass dabei die optimale Lösung aus dem Raum gelöscht wird.

Alle bisherigen Suchmethoden waren in gewisser Hinsicht naive, direkte Ansätze. Eine völlig andere Sichtweise der Suche bekommt man mit den stochastischen Suchverfahren. Hierbei werden Zufälle in den Verfeinerungsprozess mit eingebracht. Diese zufällige Auswahl geeigneter Verfeinerungen kann man auf die bisherigen Varianten anwenden. So erhält man eine

[2] Die neuen Features entsprechen Attributen, die vorher nicht in der Trainingsmenge vorhanden waren und die *dynamisch* ergänzt werden.

stochastische Hill-Climbing Suche, wenn man zufallsbasierte Generalisierungs- und Spezialisierungsoperatoren einführt, die mit einer höheren Wahrscheinlichkeit gute Regeln finden, aber schlechten Kandidaten ebenfalls noch eine Chance einräumen. Ein weiterer Ansatz wäre, die Wahrscheinlichkeit eine schlechte Regel zu selektieren über die Zeit abnehmen zu lassen, was bei *Simulated Annealing* zum Tragen kommt. Oder man verwendet genetische Algorithmen, bei denen die aktuelle Generation den jeweils gespeicherten Kandidatenregeln entspricht. Eine neue Generation erhält man durch zufälliges Generieren von neuen Regeln, durch zufälliges Generalisieren von alten Regeln oder durch zufälliges Austauschen von Bedingungen zwischen den Regeln. Die so entstandenen Regeln werden evaluiert und ein fester Teil in die nächste Generation weitergegeben. Sobald eine Regel eine bestimmte Anzahl an Generationen überlebt, wird abgebrochen und diese Regel zurückgegeben. Häufig wird beim Separate-and-Conquer Regel-Lernen allerdings eine der beiden ersten Varianten benutzt, da diese bereits gute Ergebnisse bei hoher Effizienz liefern. Die beiden letzten Methoden leiden vor allem an Ineffizienz, wobei es durchaus Szenarien gibt, bei denen eine bessere Suche vonnöten ist.

Die nächste „Steuerung“ der Suche erfolgt über die Strategie, mit der der Raum der verschiedenen Regeln durchsucht wird. Hierbei gibt es drei verschiedene Möglichkeiten. Entweder man geht von einer allgemeinen Kandidatenregel aus, die in weiteren Schritten immer mehr spezialisiert wird oder man startet mit einer sehr speziellen Regel, die häufig genau ein Beispiel umfasst, und generalisiert diese im weiteren Verlauf. Als letzte Option bleibt eine Vermischung dieser beiden Strategien, die als Top-Down- und Bottom-Up-Strategie bezeichnet werden. Unter dem Generalisieren einer Regel versteht man das Entfernen von Bedingungen, so dass die Regel danach mehr Beispiele als vorher abdeckt. Genau umgekehrt verhält es sich beim Spezialisieren. Hat man beispielsweise die Datenmenge aus Tabelle 2.1 und diese durch die beiden Regelmengen

$$R_1 = \{Wetter = sonnig \rightarrow SpieleTennis = ja\} \text{ und}$$

$$R_2 = \{Wetter = regnerisch \wedge Wind = ja \rightarrow SpieleTennis = ja\} \tag{2.5}$$

beschrieben, so werden mit R_1 zwei positive und zwei negative Beispiele (die Beispiele A,B,C und D) abgedeckt und mit R_2 kein positives und ein negatives Beispiel (das Beispiel E).

Beispiel	Wetter	Temperatur	Wind	SpieleTennis
A	sonnig	30	ja	ja
B	sonnig	30	ja	ja
C	sonnig	25	nein	nein
D	sonnig	22	nein	nein
E	regnerisch	20	ja	nein

Tabelle 2.1: Beispiel für eine Trainingsmenge

Fügt man nun der ersten Regel die Bedingung „$Temperatur > 29$“ hinzu, so werden beide positiven Beispiele abgedeckt (wie bereits vorher), aber keines der negativen Beispiele mehr. In diesem Fall hat man die Regel spezieller gemacht und somit auch weniger Beispiele abgedeckt. Umgekehrt kann man mit einer Generalisierung bei der zweiten Regel die Bedingung „$Wetter = regnerisch$“ entfernen. Man deckt nun zwei positive und ein negatives Beispiel ab, also insgesamt mehr Beispiele als vorher. Am häufigsten wird beim Separate-and-Conquer Regel-Lernen eine Top-Down Strategie verwendet, die durch das Hinzufügen von Bedingungen versucht immer mehr negative Beispiele auszuschließen.

Die letzte Möglichkeit den Algorithmus über die Suchparameter zu justieren ist die Such-Heuristik. Diese hat den größten Einfluss auf den Lernprozess. Generell wird eine Heuristik beim Separate-and-Conquer Regel-Lernen zur Bewertung der Güte einer gefundenen Kandidatenregel verwendet. Abhängig von dem Evaluationswert, der der Regel von der Heuristik zugeordnet wird, entscheidet es sich, ob sie der Theorie hinzugefügt wird. Ist der Wert noch nicht hoch genug, wird die Regel weiter verfeinert beziehungsweise generalisiert oder sogar verworfen. Die Evaluation richtet sich im Gegensatz zu den Heuristiken die im Entscheidungsbaumlernen verwendet werden ausschließlich nach den Beispielen, die von der aktuellen Regel abgedeckt werden. Da in dieser Arbeit das höchste Gewicht auf der Auswahl und der Erstellung einer geeigneten Heuristik lag, werden diese in Kapitel 3 eingehend beschrieben.

2.1.3 Methoden zur Vermeidung von Overfitting

Eine weitere Möglichkeit, die verschiedenen Separate-and-Conquer Algorithmen zu unterscheiden, ist ihre Methode zur Vermeidung von *Overfitting*. Das Problem des *Overfitting* tritt häufig bei verrauschten Daten auf. Sobald der Algorithmus versucht die fehlerhaften Daten zu beschreiben, kann es dazu kommen, dass er sich zu genau an die Trainingsmenge anpasst. Sollen nun neue ungesehene Beispiele klassifiziert werden, ist die Wahrscheinlichkeit einer fehlerhaften Klassifikation sehr hoch. Ein Algorithmus kann sich jedoch auch zu genau an eine Trainingsmenge anpassen, obwohl diese keine Fehler in den Daten enthält. Erstellt man beim Regel-Lernen beispielsweise für jedes Beispiel der Trainingsmenge genau eine Regel, so werden mit der resultierenden Menge die Daten absolut exakt beschrieben. Sobald aber ein Beispiel klassifiziert werden soll, was so in den Trainingsdaten nicht vorhanden ist, versagt der Klassifikator. Aus diesem Grund ist es wichtig, Mechanismen in den Algorithmus einzubauen, die eine solche Überangepasstheit zu verhindern wissen. Im Allgemeinen ist man versucht möglichst generelle Regeln zu finden, die eine hohe Abdeckung haben. Je allgemeiner eine Regel ist, desto niedriger ist die Wahrscheinlichkeit, dass sie auf ein ungesehenes Beispiel nicht zutrifft. *Overfitting* entsteht häufig durch komplexe Regeln, die nur wenige Beispiele abdecken. Die Abbruchbedingung eines Separate-and-Conquer

Algorithmus gibt an, wie lange versucht wird die Theorie zu verbessern und ab welchem Zeitpunkt die Regelmenge als gut genug gilt. Je nachdem welche Kriterien hier vorgegeben sind, wird durch die Abbruchbedingung das *Overfitting* verhindert. Muss beispielsweise die Bedingung erfüllt sein, dass der Algorithmus alle positiven (Vollständigkeit) und kein einziges negatives Beispiel abdecken soll (Konsistenz), so kann es leicht zu *Overfitting* kommen. Mit der Abbruchbedingung kann man steuern, wie stark man *Overfitting* vermeiden möchte.

Man unterscheidet zwei verschiedene Möglichkeiten, eine Regelmenge oder einzelne Regeln zu beschneiden: Versucht man bereits während des Lernprozesses die Regeln zu vereinfachen, spricht man vom *Pre-Pruning*. Wird dieser Schritt erst auf die fertige Theorie angewendet, so handelt es sich um *Post-Pruning*.

Pre-Pruning Es gibt beim *Pre-Pruning* verschiedene Methoden um *Overfitting* zu verhindern. So kann man zum Beispiel eine bestimmte Prozentzahl von positiven Beispielen vorgeben, die eine Regel mindestens abdecken muss. Häufig liegt diese bei 80 %. Sobald die bestmögliche Regel weniger als 80 % positive Beispiele abdeckt, wird sie verworfen und die bisher gefundene Theorie als optimal angesehen. Eine andere Möglichkeit basiert auf der *Minimum Description Length MDL* (siehe Definition 2.3), die angibt, wie viele Bits mindestens benötigt werden, um eine Regel zu beschreiben. Auch hier wird eine Grenze eingeführt, um komplizierte Regeln, die nur wenige Beispiele abdecken zu verhindern. Variiert die Verteilung von positiven und negativen Beispielen der Trainingsmenge zu der Verteilung der durch die Regel abgedeckten Beispiele nur gering, so ist dies ein Indiz dafür, dass die Regel wahrscheinlich viele negative Beispiele mit abdeckt[3]. Bei dieser Methode würde eine Regel zurückgewiesen werden, wenn der Unterschied in der Verteilung nicht signifikant ist. Eine weitere Möglichkeit wäre, nicht die gesamte Regel zu evaluieren, sondern jeden Teil aus dem die Regel zusammengesetzt ist, einzeln zu untersuchen. Auch hier kann der Benutzer für eine Bewertungsfunktion einen Schwellwert einführen, ab dem ein Literal in der Regel erhalten bleibt. Empirisch hat sich bei der Korrelations-Heuristik, die beim Algorithmus FOSSIL [13] eingesetzt wird, ein Wert von 0.3 als gut erwiesen.

Post-Pruning Beim *Post-Pruning* steht die fertige Theorie bereits zu Verfügung. Es wird nun versucht redundante Bedingungen aus den Regeln zu entfernen oder sogar ganze Regeln zu löschen. Man misst die Genauigkeit der Theorie und beginnt dann versuchsweise Bedingungen und Regeln zu entfernen. Verschlechtert sich die Genauigkeit nicht, so werden die gelöschten Teile dauerhaft entfernt. Mit *Post-Pruning* Methoden können bessere Theorien als mit *Pre-Pruning* Verfahren gefunden werden. Es wurde in [1] empirisch

[3]Da immer versucht wird möglichst ausschließlich positive Beispiele abzudecken und man approximiert von einer Gleichverteilung der Beispiele in der Trainingsmenge ausgehen kann, ist eine hohe Unterschiedlichkeit in der Verteilung ein Indiz für eine gute Regel. Je stärker die Variation in der Verteilung der positiven und negativen Beispiele ist, desto besser ist die Regel (da die Wahrscheinlichkeit von einer der 2 Klassen mehr Beispiele abzudecken größer ist).

nachgewiesen, dass der Algorithmus *REP* (Reduced Error Pruning), welcher auf einer *Post-Pruning* Strategie beruht, besser abschneidet als *FOIL* [23], der auf *Pre-Pruning* setzt. Der Nachteil liegt aber in der Ineffizienz. Eine interessante Verbesserung der *Post-Pruning* Verfahren ergibt sich, wenn man eine *Growing-and-Pruning* Strategie benutzt, wie sie beim *REP* Algorithmus verwendet wird. Hierbei wird die Trainingsmenge am Anfang in $\frac{2}{3}$ für die *Growing-Menge* und $\frac{1}{3}$ für die *Pruning-Menge* aufgespalten. Auf der *Growing-Menge* wird wie gehabt gelernt, während alle Evaluationen, die für die Pruning Phase nötig sind, auf der *Pruning-Menge* errechnet werden. Eine ausführliche Erklärung der *Growing-and-Pruning* Strategie wird in Abschnitt 2.4 bei *JRip* gegeben.

Um in der *Post-Pruning* Phase weniger Arbeit zu haben, bietet sich eine Kombination der beiden Pruning-Strategien an. So wird während des Lernprozesses bereits geprunt und die fertige, vereinfachte Theorie nochmals in einer *Post-Pruning* Phase angepasst. Ein daraus resultierendes Problem liegt beim Regel-Lernen darin, dass durch das Entfernen bestimmter Bedingungen einer Regel andere Regeln beeinflusst werden. Durch das Wegschneiden von Bedingungen wird die Regel generalisiert und deckt mehr Beispiele ab. Kommt es nun zu der Situation, dass die geprunte Regel alle Beispiele, die durch eine spätere Regel klassifiziert wurden, abdeckt, so wird die spätere Regel in der *Post-Pruning* Phase komplett entfernt, da bereits alle Beispiele von der ersten Regel abgedeckt werden. Wird hingegen nur ein gewisser Teil der Beispiele von der geprunten Regel abgedeckt, so ist es unmöglich durch Generalisierung der folgenden Regel den Teil der Beispiele, der nicht durch die erste Regel abgedeckt wird, mit möglichst wenigen Bedingungen zu beschreiben. Dieses Problem wird durch ein iteratives Verfahren in den Griff bekommen auf das ebenfalls in Abschnitt 2.4 näher eingegangen wird.

2.2 Separate-and-Conquer Regel-Lernen

Im vorherigen Abschnitt wurden die Unterschiede zwischen den verschiedenen Separate-and-Conquer Algorithmen aufgezeigt. Nun soll die Strategie, die diesen Algorithmen zugrunde liegt, näher erläutert werden.

2.2.1 Die Separate-and-Conquer Strategie

Separate-and-Conquer Algorithmen haben ihren Ursprung in der Familie der AQ-Algorithmen [19]. Hier wird diese Vorgehensweise als Covering-Strategie bezeichnet. Den Begriff Separate-and-Conquer haben Pagallo und Haussler [21] eingeführt. Das Ziel dieser Familie von Algorithmen ist es, die positiven Beispiele der Trainingsmenge durch eine Menge von Regeln abzudecken, so dass für jedes positive Beispiel mindestens eine Regel zutrifft. Feuert

keine der Regeln bei einem Beispiel, so wird es automatisch negativ klassifiziert. Die Vorgehensweise dieser Algorithmen lässt sich in einen „Abtrennungsschritt“ *(Separate)* und einen „Eroberungsschritt“ *(Conquer)* aufspalten. In einer inneren Schleife wird eine Regel gesucht, die einen Teil der positiven Beispiele in den Trainingsdaten abdeckt. Sobald eine solche Regel gefunden ist, wird diese der Theorie hinzugefügt. Nun wurde dieser Teil der Trainingsmenge „erobert“. Daraufhin werden alle Beispiele[4], die diese Regel abdeckt, aus der Trainingsmenge entfernt, bzw. „abgetrennt“. Dieser „Abtrennungsschritt“, der im Algorithmus durch eine äußere Schleife realisiert wird, ist bei allen Separate-and-Conquer Algorithmen gleich. Die im Abschnitt 2.1 aufgezeigten Unterschiede zwischen den Algorithmen werden in der inneren Schleife implementiert. Je nachdem nach welchem Mechanismus man hier geeignete Regeln findet, ergeben sich die verschiedenen Möglichkeiten, die Regeln zu repräsentieren, die Suche zu instantiieren und *Overfitting* zu vermeiden. Letzteres kann zusätzlich in der äußeren Schleife durch ein bestimmtes Abbruchkriterium und eine Nachbearbeitung der fertigen Theorie erreicht werden. Hier könnte man beispielsweise das Abbruchkriterium verschwächen, so dass nicht mehr alle positiven Beispiele abgedeckt sein müssen, sondern nur ein gewisser Prozentsatz.

2.2.2 Ein generalisierter Separate-and-Conquer Algorithmus

Ein generalisierter Separate-and-Conquer Algorithmus wurde in [7] beschrieben. Die Implementation ist hier kurz skizziert. Das generalisierte Schema zum Separate-and-Conquer Regel-Lernen gliedert sich in zwei Prozeduren. In der ersten Prozedur (**SeparateAndConquer**) wird eine äußere Schleife (der Separate-Schritt) implementiert. Diese läuft so lang, bis das *RuleStoppingCriterion* zutrifft. In der Schleife wird in jedem Iterationsschritt eine beste Regel mit der zweiten Prozedur **FindBestRule** gesucht (der Conquer-Schritt). Die Beispiele, die die gefundene Regel abgedeckt, werden aus der Trainingsmenge entfernt. Die Regel wird der Theorie hinzugefügt und es wird von vorne begonnen. Als letztes wird auf die gefundene Theorie gegebenenfalls noch ein Post-Pruning angewendet, welches in der Methode *PostProcess* implementiert ist.

In der Prozedur **FindBestRule** wird versucht eine möglichst gute Regel zu finden. Die Güte einer Regel orientiert sich an einem numerischen Wert, der durch eine Heuristik bestimmt wird. Die Methode *EvaluateRule* ist für die Bestimmung dieses Wertes zuständig. Auf die speziellen Eigenschaften einer guten Heuristik wird in Kapitel 3 genauer eingegangen. Zu Beginn wird eine Anfangsregel initialisiert. Das Paar *<Evaluationswert, Regel>* wird in einer sortierten Liste (*rules*) gespeichert und die Suche wird begonnen. Solange die

[4] Man unterscheidet Algorithmen, die nur die positiven Beispiele entfernen von denen, die alle Instanzen aus der Trainingsmenge löschen.

Algorithm 1 Der generalisierte Separate-and-Conquer Algorithmus

```
PROCEDURE SeparateAndConquer (examples)
{
  theory = ∅
  WHILE Positive(examples) ≠ ∅
  {
    rule = FindBestRule (examples)
    covered = Cover (rule, examples)
    IF (RuleStoppingCriterion (theory, rule, examples))
    {
      EXIT WHILE
    }
    examples = examples \ covered
    theory = theory ∪ rule
  }
  theory = PostProcess (theory)
  RETURN (theory)
}
PROCEDURE FindBestRule (examples)
{
  initRule = InitializeRule (examples)
  initVal = EvaluateRule (initRule)
  bestRule = <initVal, initRule>
  rules = {bestRule}
  WHILE Rules ≠ ∅
  {
    candidates = SelectCandidates (rules, examples)
    rules = rules \ candidates
    FOREACH candidate IN candidates
    {
      refinements = RefineRule (candidate, examples)
      FOREACH refinement IN refinements
      {
        evaluation = EvaluateRule (refinement, examples)
        WHILE (!StoppingCriterion (refinement, evaluation, examples)
        {
          newRule = <evaluation, refinement>
          rules = InsertSort (newRule, rules)
          IF (newRule > bestRule)
          {
            bestRule = newRule
          }
        }
      }
    rules = FilterRules (rules, examples)
  }
  RETURN (bestRule)
}
```

Liste *rules* nicht leer ist, werden aus ihr Kandidatenregeln in eine weitere Liste *candidates* verschoben. Diese Kandidatenregeln werden nun verfeinert. Dabei kann in der Methode *RefineRule* angeben werden, ob eine Top-Down- oder eine Bottom-Up-Strategie zu verwenden ist. Jede Verfeinerung wird nun evaluiert und von dem *StoppingCriterion* überprüft. Trifft es zu, so wird der Verfeinerungsprozess unterbrochen, da eine weitere Verbesserung der Regel nicht mehr möglich ist. Ansonsten wird die verfeinerte Regel in die Liste *rules* einsortiert. Ist nun die neu gefundene Regel höher evaluiert worden als die bisher beste,

so gilt sie im folgenden als beste Regel. Zum Schluss wird mit der Methode *FilterRules* entschieden, welche der gespeicherten Regeln in die nächste Iteration übergeben werden und die beste Regel wird zurückgegeben.

Mit diesem generellen Schema können alle Separate-and-Conquer Regel-Lerner implementiert werden, je nachdem welche Funktionalität man den einzelnen Methoden einimpft. Über die Methode *RefineRule* kann man beispielsweise justieren, aus welchen Bedingungen eine Regel überhaupt besteht. Durch eine Festlegung der Gestalt der Bedingungen (beispielsweise *DNF*) kann die Repräsentationssprache unterschiedlich definiert werden. Den Suchalgorithmus kann man über *SelectCandidates* und *FilterRules* steuern, indem man beispielsweise zur Initialisierung von Hill-Climbing in *FilterRules* festlegt, dass nur die beste Regel in die nächste Iteration gelangt. In diesem Fall kann in *SelectCandidates* nur eine Regel ausgewählt werden. Die Suchstrategie ist über die Methoden *InitializeRule* und *RefineRule* steuerbar. So würde man zur Initialisierung einer Top-Down-Strategie die generellste Regel verwenden und diese dann in *RefineRule* weiter spezialisieren. Die Suchheuristik ist in *EvaluateRule* implementiert. Schließlich stehen zur Vermeidung von *Overfitting* die beiden Abbruchkriterien oder die Post-Process Phase zu Verfügung.

2.2.3 Die Sortierung der Regeln in der Liste

Nicht unerheblich ist beim Regel-Lernen die Sortierung der Kandidatenregeln innerhalb der Liste *rules*. Im Unterschied zu Entscheidungsbäumen ist es bei Regeln möglich, dass sich einzelne Bedingungen überlappen, also in mehreren Regeln vorkommen, die unterschiedliche Klassenvorhersagen treffen[5]. So kann durch eine Umsortierung der Regeln in der Liste eine signifikant andere Klassifikation einzelner Beispiele entstehen.

Üblicherweise arbeiten Separate-and-Conquer Algorithmen intern immer mit Zwei-Klassen-Problemen. Das Zielkonzept soll zwischen positiven und negativen Beispielen unterscheiden können. Handelt es sich um ein positives Beispiel, so trifft eine der vorhandenen Regeln zu. Ist das Beispiel negativ, so feuert keine der Regeln, was gleichzusetzen ist mit einer einzigen Regel für alle negativen Beispiele *(negation as failure)*. Bei 2-Klassen-Problemen ist die Sortierung der Regeln beliebig, da ein Beispiel als positiv klassifiziert wird, sobald es abgedeckt ist. An welcher Position die zutreffende Regel in der Liste steht, ist hierbei unerheblich. In der Realität beschäftigt man sich aber meist mit Mehr-Klassen-Problemen. Wird ein Beispiel von mehreren Regeln abgedeckt, die aber unterschiedliche Vorhersagen treffen, wird die Ordnung der Regeln innerhalb der Liste wichtig. Da sie von oben nach unten abgearbeitet wird und die erste Regel, die feuert, den Klassenwert angibt, ist die Reihenfolge bei dieser Art von Problemen von höchster Bedeutung.

[5] Einzelne Äste können sich bei Entscheidungsbäumen nicht überlappen.

Es gibt verschiedene Konzepte Regel-Lernern zu ermöglichen, mit Mehr-Klassen-Problemen umzugehen. Eines ist die Einführung von *homogenen Regeln* [27], die die Eigenschaft besitzen, dass alle Verfeinerungen der Regel denselben Evaluationswert besitzen. Da jede Entscheidungsliste in eine logisch äquivalente homogene Entscheidungsliste konvertierbar ist, wie ebenfalls in [27] gezeigt wurde, ist bei jener die Sortierung innerhalb der Liste nicht entscheidend. Eine andere Möglichkeit, die im Algorithmus *CN2* implementiert wurde, ist die Fokussierung auf Bereiche der Beispielmenge, wo eine Klasse die Verteilung dominiert. Um zu verhindern, dass dennoch mehrere Regeln feuern, werden die Regeln in der Reihenfolge wie sie gefunden wurden in die Liste einsortiert. Ein weiterer Ansatz wäre, für jede mögliche Klasse ein eigenes Zielkonzept zu lernen, wobei die zugehörigen Beispiele jeweils als positiv definiert werden und alle verbleibenden als negativ. Zusätzlich wird mit einer geeigneten Heuristik jeder Regel ein Gewicht zugeordnet. Neue Beispiele werden dann von der Regel mit dem höchsten Gewicht der entsprechenden Klasse zugeordnet. Diese Methode wird häufig bei Separate-and-Conquer Algorithmen verwendet.

2.3 Die Rolle der Heuristiken beim Regel-Lernen

Den größten Einfluss auf die Regeln, die der Theorie hinzugefügt werden, hat die Heuristik eines Regel-Lern Algorithmus. Durch die Messung der Güte einer Regel wird ein Vergleich von unterschiedlichen Regeln erst ermöglicht. Je nachdem wie diese Messung durchgeführt wird, ergeben sich andere Präferenzen und eine andere Steuerung des Algorithmus durch den Raum der Kandidatenregeln. Momentan wird im Regel-Lernen versucht, die Güte einer Regel danach abzuschätzen, wie viele positive und negative Beispiele von dieser abgedeckt werden. Andere Möglichkeiten könnten hier vielversprechender sein. So wurde zum Beispiel in [8] versucht die Güte einer Kandidatenregel nach der Wahrscheinlichkeit zu beurteilen, dass sie zu einer Regel verfeinert wird, die in der resultierenden Theorie vorkommt. Das Problem einer Messung der von der Regel abgedeckten positiven und negativen Beispiele besteht darin, dass sie immer auf der Trainingsmenge erfolgt. Das Ergebnis wird hier zu optimistisch ausfallen, da die Testmenge häufig anders beschaffen ist, also beispielsweise eine unterschiedliche Klassenverteilung aufweist. In [8] wurde daher versucht, die Genauigkeit einer Regel auf der Testmenge vorherzusagen. Dieser Ansatz ist Erfolg versprechend und begegnet dem Problem der zu optimistischen Abschätzung von Regeln.

In dieser Arbeit liegt das Hauptaugenmerk auf Heuristiken, die die Parameter aus Tabelle 2.2 erhalten und die Güte der Regeln auf der Trainingsmenge ermitteln. Vorgänger einer Regel (also die vorherige Verfeinerung) werden zur Berechnung des Evaluationswertes nicht weiter beachtet.

p	die positiven Beispiele aus der Trainingsmenge, die von der Regel abgedeckt werden
n	die negativen Beispiele aus der Trainingsmenge, die von der Regel abgedeckt werden
P	die Anzahl der positiven Beispiele insgesamt in der Trainingsmenge $(TP + FN)$
N	die Anzahl der negativen Beispiele insgesamt in der Trainingsmenge $(FP + TN)$
TPR	die TruePositiveRate $\frac{p}{P}$, die Anzahl der positiven Beispiele von allen positiven Beispielen der Trainingsmenge
FPR	die FalsePositiveRate $\frac{n}{N}$, die Anzahl der negativen Beispiele von allen negativen Beispielen der Trainingsmenge
TP	TruePositive, ein positives Beispiel, welches positiv vorhergesagt wurde, also richtig klassifiziert wurde *(p)*
FN	FalseNegative, ein positives Beispiel, welches negativ vorhergesagt wurde, also falsch klassifiziert wurde *(P-p)*
FP	FalsePositive, ein negatives Beispiel, welches positiv vorhergesagt wurde, also falsch klassifiziert wurde *(n)*
TN	TrueNegative, ein negatives Beispiel, welches negativ vorhergesagt wurde, also richtig klassifiziert wurde *(N-n)*
$total$	die Gesamtanzahl von Beispielen in der Trainingsmenge $(P + N = TP + FN + FP + TN)$

Tabelle 2.2: Notation

Definition 2.2 (Heuristik) Eine Heuristik *h(R)* ist eine Funktion, die als Eingabeparameter eine Regel R erhält und einen Wert ausgibt, der die Güte der Regel R beschreibt. Im Folgenden gilt: $h(R) \equiv h$. Die Begriffe *Heuristik* und *Maß* sind als äquivalent anzusehen.

Es gibt verschiedene Heuristiken, deren Darstellungen als Isometriken variieren. Eine Unterscheidung der Basis-Modelle wird in Abschnitt 3.1 vorgenommen. Dort wird auch beschrieben, welche Konzepte einer guten Heuristik zugrunde liegen. Die parametrisierbaren Heuristiken werden in diesem Abschnitt ebenfalls detailliert erläutert.

2.4 JRip

JRip ist eine im *weka-Framework* [33] implementierte Variante von *RIPPER* [3]. Sie beruht auf dem Konzept des *Incremental Reduced Error Pruning (IREP)* [12]. Hierbei handelt es sich um eine Kombination des Separate-and-Conquer Prinzips mit einer Growing-and-Pruning Strategie. Eine Schleife läuft solange über alle Beispiele der Trainingsmenge, bis alle positiven abgedeckt sind und sucht eine gute Regel *(conquer)*. Sobald diese gefunden ist, werden die abgedeckten Beispiele aus der Trainingsmenge entfernt und die nächste Regel auf den verbleibenden Beispielen gesucht *(separate)*. Die Suche nach einer guten Regel

funktioniert, indem die Trainingsmenge in eine Growing-Menge (hier: $\frac{2}{3}$) und eine Pruning-Menge ($\frac{1}{3}$) aufgeteilt wird. Auf der Growing-Menge wird eine Regelmenge gefunden, die stark an diese angepasst ist (die Growing-Menge möglichst overfitted). Dann werden sukzessive Bedingungen der Regeln entfernt und ihre Fehlerrate auf der Pruning-Menge gemessen. In der Implementierung von *IREP* wurde die Regel der Theorie hinzugefügt, solange die Fehlerrate unter 50 % lag. Es ist aber in [3] gezeigt worden, dass dieses Abbruchkriterium *IREP* dazu bringt, frühzeitig zu stoppen. Mit dem Kriterium ist *IREP* anfällig für das *Small Disjunct Problem* [15]. Small Disjuncts sind sehr komplexe Regeln, die wenige positive Beispiele der Trainingsmenge abdecken. Diese besitzen auf der Testmenge häufig eine hohe Fehlerrate, sind aber dennoch nötig, um eine vollständige Theorie zu erreichen. Daher besteht bei *IREP* das Problem, dass die Wahrscheinlichkeit hoch ist, in einer Serie von Regeln, die auf der Growing-Menge nur wenige Beispiele abdecken, eine Fehlerrate auf der Pruning-Menge zu erreichen, die über 50 % liegt. Aus diesem Grund wird in der Implementierung von *RIPPER* ein anderes Konzept verwendet, um ein besseres Abbrechen des Hinzufügens einzelner Regeln zu definieren.

Definition 2.3 (Minimum Description Length) $MDL(H) = I(H) + I(E|H)$, wobei *I(H)* angibt, wie groß die Menge an Information ist, die benötigt wird, um die Hypothese H zu übertragen und *I(E|H)* die Menge an Information, die zur Übermittlung der Beispielmenge E nötig ist, wenn man die Hypothese H zur Hilfe nehmen kann (es wird zuerst die Hypothese übertragen und dann die Beispiele, deren Übertragungsvolumen sich durch die Hypothese verringert)

Eine genaue Beschreibung dieses Prinzips ist in [32] zu finden.

RIPPER misst nun die *MDL* nach jedem Hinzufügen einer neuen Regel. Sobald die neue *MDL d* bits größer als die kleinste bisherige ist, wird das Suchen neuer Regeln unterbrochen. Als guter Wert hat sich *d=64* ergeben.

Als Algorithmus, der auf der Growing-Menge die Regeln findet, wird *FOIL* [23] verwendet. Begonnen wir mit einer leeren Initialregel, der Bedingungen der Form

$$A_n = v,\ A_k \leq x \text{ oder } A_k \geq x \text{ mit} \tag{2.6}$$

A_n= *nominales Attribut,* v = *zulässiger Wert von* A_n *und*

A_k= *kontinuierliches Attribut,* x = *ein Wert von* A_k*, der in den Trainingsdaten vorkommt*

hinzugefügt werden können. Es werden die Bedingungen ausgewählt, die *Weighted Information Gain WIG* (Definition 2.6) maximieren.

Mit Information Content (Informationsgehalt) wird gemessen, wie viele Informationen in der Klassifikation der abgedeckten Beispiele enthalten ist.

Definition 2.4 (Information Content) $IC(R) = -log\frac{p}{p+n}$, wobei mit R die zu bewertende Regel gemeint ist

Definition 2.5 (Positive Abdeckung) $C(R) = p$

Dieses Konzept ist äquivalent zum Recall und wird in Abschnitt 3.3.2 näher beschrieben.

Definition 2.6 (Weighted Information Gain) $WIG(R) = -C(R) * (IC(R) - IC(R'))$, wobei mit R' der Vorgänger der Regel R gemeint ist

Bei Weighted Information Gain wird die Differenz zwischen dem Informationsgehalt einer Regel zu dem der nächsten Verfeinerung der Regel gebildet. Dieser wird mit der Anzahl der abgedeckten positiven Beispiele gewichtet. Hier soll sich nicht weiter mit dem Maß zur Bestimmung einer guten Bedingung beschäftigt werden, da es sich um die Optimierung einer Verfeinerung handelt und nicht um die einer einzelnen Regel. Aus diesem Grund wurde *WIG* auch nicht als Heuristik in der Arbeit verwendet. Genauer wird die Optimierung einer Verfeinerung mit Hilfe des Informationsgehaltes in [7] beschrieben.

Ist nun eine Regel erstellt worden, wird diese sofort auf der Pruning-Menge wieder vereinfacht. Gelöscht werden kann in der Implementierung von *RIPPER* eine beliebige fertige Sequenz von Bedingungen einer Regel. Verwendet wird die Vereinfachung, die mit der Heuristik

$$h_{RIPPER} = \frac{p-n}{p+n} \tag{2.7}$$

den höchsten Wert auf der Pruning-Menge erreicht.

Zusätzlich zu den Verbesserungen bei der Evaluationsheuristik einzelner Regeln und dem Abbruchkriterium ist bei *RIPPER* noch eine PostPass Phase implementiert worden, welche eine Regelmenge nach einem einfachen *REP* Prinzip (also nicht-inkrementell) versucht zu optimieren. Als erstes wird mit dem Algorithmus, in dem bereits die beiden oben beschriebenen Verbesserungen implementiert sind, eine Initial-Regelmenge $\{R_1, R_2, ..., R_k\}$ gefunden. Für jede Regel R_i dieser Theorie werden der Reihe nach 2 Alternativen erstellt. Die erste wird *Replacement* von R_i genannt und wird auf der Growing-Menge gefunden. Dann wird versucht auf der Pruning-Menge den Fehler der Regelmenge $\{R_1, ..., R_i', ..., R_k\}$ zu minimieren. Bei der zweiten, der *Revision* von R_i, wir analog vorgegangen, nur dass die Regel R_i weiter verfeinert wird (und nicht eine komplett neue Regel erstellt wird). Es wird danach basierend auf *MDL* entschieden, ob die ursprüngliche Regel R_i, das *Replacement* von R_i oder die *Revision* von R_i der Theorie hinzugefügt wird. Verbleiben in der resultierenden Regelmenge positive Beispiele, so wird der gesamte Algorithmus auf diese erneut angewendet. Dieser Mechanismus wird als *RIPPER* (Repeated Incremental Pruning to Produce Error Reduction) bezeichnet.

Abschließend sei noch kurz auf die Güte dieses Algorithmus verwiesen. Er gewinnt auf 20 von 37 Datenmengen gegen *C4.5rules* [26], erreicht 15 Unentschieden und verliert lediglich zwei mal. Diese Ergebnisse werden erreicht obwohl sich die Effizienz im Vergleich zu *IREP* nicht verschlechtert hat und *RIPPER* auf großen Datenmengen (ca. 500000 Beispiele) ungefähr sieben Minuten läuft, während *C4.5rules* ganze 79 Jahre benötigen würde [3].

2.5 ROC-Raum

Zur Evaluation von kompletten Theorien (fertigen Klassifikatoren), von einzelnen Teilen eines Konzeptes (zum Beispiel einer einzelnen Regel) oder von Ranking-Algorithmen eignet sich der ROC-Raum [6]. ROC steht für Receiver Operating Characteristics oder auch für Receiver Operating Curve und hat seine Ursprünge in der Signaltheorie.

Üblicherweise errechnet man die Genauigkeit eines Klassifikators anhand der folgenden Konfusionsmatrix, die auch als Kreuztabelle oder in [6] als Kontingenztabelle bezeichnet wird.

	positiv vorhergesagt (von einer Regelmenge abgedeckt)	negativ vorhergesagt (von einer Regelmenge nicht abgedeckt)	
tatsächlich positiv	*p (TP)*	*P-p (FN)*	*P*
tatsächlich negativ	*n (FP)*	*N-n (TN)*	*N*
	p+n	*P+N-(p+n)*	*total=P+N*

Tabelle 2.3: 2x2 - Konfusionsmatrix

Die Fehlerrate eines Klassifikators wird beispielsweise wie folgt berechnet:

Definition 2.7 (Fehlerrate) $Fehlerrate = \frac{FP+FN}{TP+FP+FN+TN} = \frac{n+P-p}{P+N}$

Daraus kann man dann auch die Genauigkeit berechnen:

Definition 2.8 (Genauigkeit über Fehlerrate) $Genauigkeit = 1 - Fehlerrate.$

Misst man allerdings die Genauigkeit eines Klassifikators mit den Parametern aus der Tabelle, so wird diese immer abhängig von der Verteilung der positiven und negativen Beispiele sein. Kommen in einer Testmenge zum Beispiel nur eine negative Instanz und tausende positive Instanzen vor, so würde man einen Klassifikator bevorzugen, der einfach immer die positive Klasse ausgibt, da er eine hohe Genauigkeit hätte. Sobald sich die Verteilung umkehrt, geht die Genauigkeit des gleichen Klassifikators gegen null. Um der Abhängigkeit

der Genauigkeit des Klassifikators von der Klassenverteilung in der Trainingsmenge zu entkommen, bewertet man bei ROC-Räumen die Raten der positiv und negativ klassifizierten Beispiele. Daher können Punkte im ROC-Raum, die eine unterschiedliche *TPR* und *FPR* aufweisen, trotzdem die gleiche Genauigkeit erhalten.

Definition 2.9 (ROC-Raum) Ein ROC-Raum wird durch die *FPR* und die *TPR* aufgespannt. Jeder Punkt (x, y) entspricht entweder einem Klassifikator oder dem Teil einer Theorie (zum Beispiel einer einzelnen Regel), die eine *FPR* von x und eine *TPR* von y haben.

Ein gutes Beispiel, um einen Überblick über einen ROC-Raum zu bekommen, ist die Grafik 2.1 aus [6].

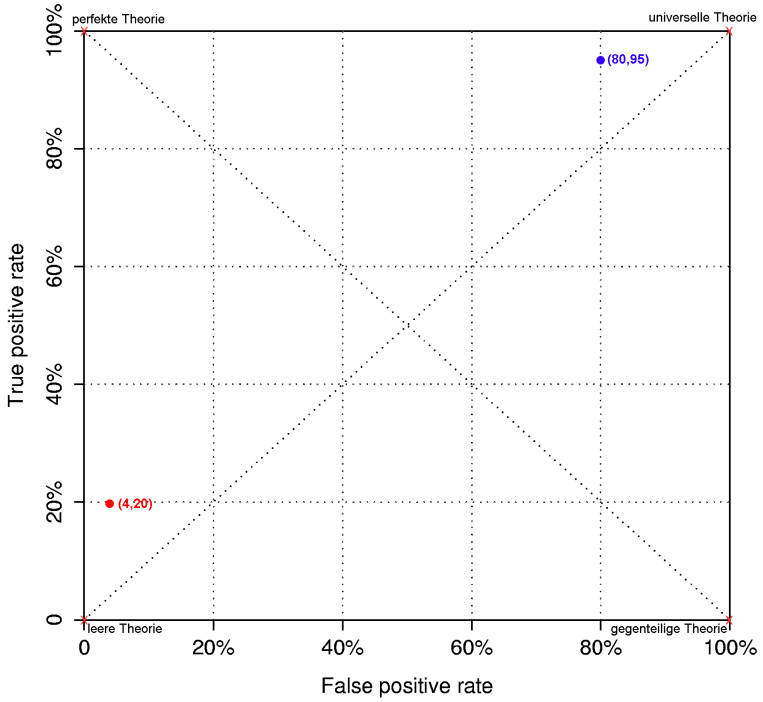

Abbildung 2.1: ROC-Raum

Zusätzlich zu den in der Grafik erkennbaren Eckdaten gilt für die beiden Diagonalen des ROC-Raumes:

- die Diagonale, die die Punkte $(0, 0)$ und $(1, 1)$ verbindet, entspricht einer zufälligen Klassifikation der Beispiele (alle Klassifikatoren im linken Dreieck der Diagonalen sind besser als eine zufällige Klassifikation und alle im rechten schlechter) und repräsentiert eine Gleichverteilung von vorhergesagten positiven und negativen Beispielen (es gilt $x{=}y$)

- die Diagonale, die die Punkte $(1, 0)$ und $(0, 1)$ verbindet, entspricht Klassifikatoren, die gleich gut auf beiden Klassen abschneiden (alle Klassifikatoren im linken Dreieck der Diagonalen schneiden besser bei den negativen Beispielen ab und alle im rechten besser auf den positiven)

Der untere Punkt in der Grafik entspricht einem diskreten Klassifikator[6], der eine *TPR* von 20 % und eine *FPR* von 4 % hat. Er gilt als konservativ [4], da er positive Vorhersagen nur mit hoher Sicherheit macht. Aus diesem Grund klassifiziert er zwar die vorhergesagten positiven Beispiele korrekt, hat also eine sehr niedrige *FPR,* deckt aber nur einen geringen Teil der gesamten positiven Beispiele ab. Der obere Punkt hingegen repäsentiert einen Klassifikator, der darauf abzielt, möglichst viele positive Beispiele korrekt vorherzusagen, der dabei aber auch viele Instanzen falsch klassifiziert. Er gilt daher als weniger konservativ.

Im Folgenden werden einige interessante Eigenschaften von ROC-Räumen näher erläutert. Üblicherweise ist das rechte Dreieck unterhalb der Diagonalen eines ROC-Raumes leer, da Klassifikatoren, die sich dort befinden, schlechter als eine zufällige Klassifikation sind. Negiert man aber die zugrunde liegende Theorie, macht also die negative zur positiven Klasse, so wird der Punkt an der Diagonalen gespiegelt, und man erhält einen Klassifikator, dessen Güte besser als eine zufällige Vorhersage ist. Generell versucht man immer, sich mit einem Klassifikator in Richtung der oberen, linken Ecke (der perfekten Theorie) zu bewegen. Mit die wichtigste Eigenschaft von ROC-Räumen ist ihre Unabhängigkeit von der Klassenverteilung und den Kosten einer fehlerhaften Klassifikation.

Definition 2.10 (Kosten) Wird es unterschiedlich bewertet, ein positives oder ein negatives Beispiel fehlerhaft zu klassifizieren, spricht man von verschiedenen Kosten. Der Profit für das richtige Klassifizieren eines Beispiels und die Kosten einer fehlerhaften Klassifikation sind nicht mehr gleich. Die Kosten kann man sich als unterschiedliche Gewichtung der Beispiele vorstellen.

Möchte man beispielsweise herausfinden, ob ein Patient an einer Seuche erkrankt ist, so wäre ein Nichterkennen der Krankheit (*FN*) wesentlich schlimmer, als wenn man die Seuche fälschlicherweise diagnostizieren würde (*FP*). Daraus folgt, dass die Kosten der fehlerhaften Klassifikation eines positiven Beispiels (der Patient ist von der Seuche befallen und wird als gesund eingeschätzt) deutlich höher sind, als der Profit einer positiven Zuordnung (die Seuche wird diagnostiziert). Häufig wird kein Profit für eine korrekte Klassifikation (*TP* und *TN*) und gewisse Kosten für eine fehlerhafte (*FP* und *FN*) angenommen.

Die Kosten einer fehlerhaften Klassifizierung stehen in Zusammenhang mit der Klassenverteilung in der Datenmenge. Hat ein Lernalgorithmus keine Methodik eingebaut, um

[6] Ein diskreter Klassifikator gibt eine eindeutige Klassenvorhersage zurück, während ein kontinuierlicher Klassifizierer (auch Ranker genannt) die Wahrscheinlichkeit für eine bestimmte Klasse errechnet.

mit einer Gewichtung der Beispiele umgehen zu können, kann man die gleiche Situation durch eine Veränderung der Klassenverteilung erreichen. Nimmt man beispielsweise keinen Profit für eine korrekte und die Kosten c für eine inkorrekte Klassifizierung an, so könnte man durch eine Erhöhung der negativen Beispiele in der Trainingsmenge um den Faktor c dieselbe Sachlage schaffen. Bei der Analyse mit ROC-Räumen geht man von unterschiedlichen Klassenverteilungen anstelle von Kosten aus [5]. Diese Annahme soll auch für die folgenden Kapitel gelten. Es wird also nicht mehr zwischen den Kosten einer fehlerhaften Klassifizierung und der Klassenverteilung in der Datenmenge unterschieden.

Es kann nun in einem ROC-Raum für unterschiedliche Vorgaben jeweils ermittelt werden, welcher Klassifikator sich für diese Situation am besten eignet. Allerdings resultiert aus der Prämisse eine Verteilung im Vorhinein festzulegen, dass nicht sofort ersichtlich ist, welcher Klassifikator der beste ist, wenn man eben keine Vorgabe hat. Auf dieses Problem wird im Abschnitt zur Evaluation von Klassifikatoren näher eingegangen.

2.5.1 Die Analyse von Ranking-Algorithmen

Ein Ranking-Algorithmus oder auch ein Ranker gibt keinen diskreten Klassenwert für ein Beispiel zurück, sondern einen numerischen Wert für beide Klassen. Er kann leicht durch Setzung eines Schwellwertes, ab dem eine Instanz der positiven Klasse zugeordnet wird, zu einem Klassifikator umgeformt werden. Für ein 2-Klassen-Problem ist es möglich ein Ranking der Beispiele zu erstellen. Das Beispiel, das mit der höchsten Wahrscheinlichkeit positiv klassifiziert werden würde, steht ganz oben. An der letzten Position des Rankings steht dann das Beispiel, welches am unwahrscheinlichsten positiv klassifiziert werden würde. Für einen Ranker kann eine ROC-Kurve gezeichnet werden, indem für jeden möglichen Schwellwert ein Punkt im ROC-Raum gebildet wird und die einzelnen Punkte dann verbunden werden. Angenommen es gibt folgende Instanzen in der Testmenge:

Instanz	Klasse	Wahrscheinlichkeit
A	n	0.9
B	p	0.7
C	p	0.6
D	n	0.2

Tabelle 2.4: Beispielhafte Instanzen für ein Ranking

Es resultieren vier Punkte im ROC-Raum. Der erste liegt bei $(0.5, 0)$ und bekommt den höchsten Schwellwert mit 0.9 zugeordnet. Die nächsten Punkte sind $(0.5, 0.5)$ mit 0.7, $(0.5, 1)$ mit 0.4 und $(1, 1)$ mit 0.2. Verbindet man nun ausgehend vom Ursprung alle Punkte miteinander, erhält man die ROC-Kurve. Man kann zur Verbesserung eines solchen

Algorithmus versuchen, die Fläche unter dieser ROC-Kurve (Area under the ROC Curve - AUC) zu optimieren. In dieser Arbeit soll aber auf Ranker nicht näher eingegangen werden. Dem interessierten Leser sei das Tutorial von Peter Flach aus [5] empfohlen.

2.5.2 Die Evaluation von Klassifikatoren

Jeder diskrete Klassifikator entspricht im ROC-Raum einem Punkt (x, y), der die *FPR* und die *TPR* der Theorie angibt. Die Punkte $(0, 0)$ (leere Theorie) und $(1, 1)$ (universelle Theorie) gelten ebenfalls als Klassifikatoren. Verbindet man nun ausgehend von der leeren Theorie die Klassifikatoren miteinander, die am nahesten an der optimalen Theorie liegen[7], bis hin zur universellen Theorie, so erhält man die *konvexe Hülle*.

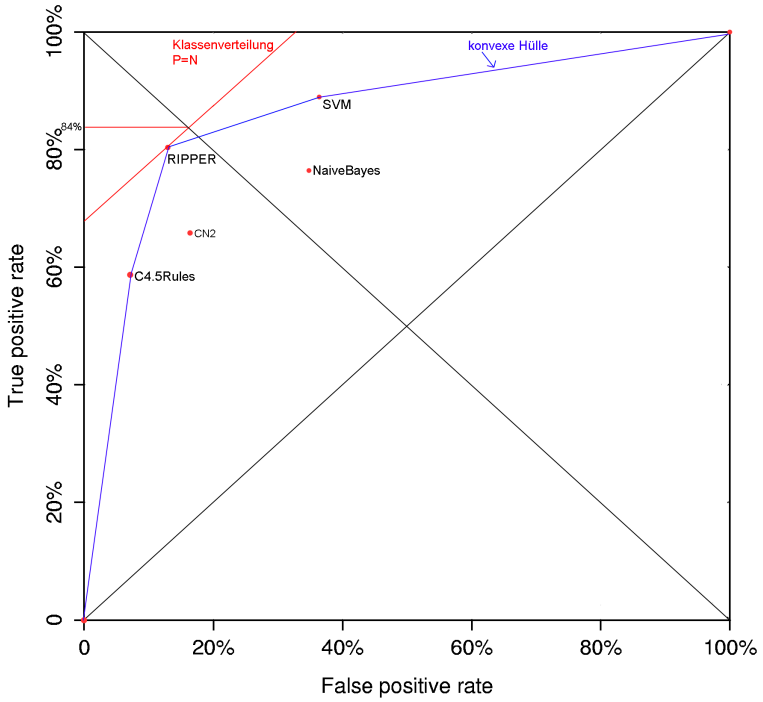

Abbildung 2.2: Konvexe Hülle und Selektion eines guten Klassifikators im ROC-Raum

Die eingezeichnete Linie, die die Klassifikatoren miteinander verbindet, entspricht dieser konvexen Hülle für die gewählte Anordnung. Alle Klassifikatoren, die unterhalb der konvexen Hülle liegen (in diesem Fall *CN2* und *NaiveBayes*) schneiden immer schlechter ab und sind daher nicht optimal. Hat man keine Vorgaben über die Klassenverteilung, gelten alle Klassifikatoren auf der konvexen Hülle als potenziell optimal. Kann man aber eine Verteilung vorgeben, so wird dies durch eine Linie im ROC-Raum repräsentiert. Kommen beispielsweise doppelt so viele positive Beispiele wie negative in der Menge vor, so erhält

[7] Die Klassifikatoren die am weitesten nordwestlich liegen.

man eine Linie mit der Steigung 0.5. Sind die negativen doppelt so häufig vertreten, resultiert eine Linie mit der Steigung zwei. Zur Ermittlung des besten Klassifikators beginnt man nun mit der Linie, die die gewünschte Verteilung repräsentiert, am Punkt $(0, 1)$ und schiebt sie entlang der Diagonalen in Richtung des Punktes $(1, 0)$. Sobald ein Klassifikator auf der Linie liegt, gilt er für diese Verteilung als optimal. Der beste Klassifikator hängt daher immer von der Verteilung der positiven und negativen Beispiele ab. Deutlich wird dies, wenn man zur Bestimmung der Güte eines Klassifikators die Genauigkeit benutzt. Angenommen die Verteilung der Beispiele ist gleich und zwei Klassifikatoren haben dieselbe Genauigkeit auf der Trainingsmenge, also klassifizieren beispielsweise 75 % der Instanzen korrekt. Der erste klassifiziert 25 von 50 positiven und alle 50 negativen Beispiele richtig, während der zweite alle 50 positiven und 25 der 50 negativen korrekt beurteilt. Je nachdem wie die Verteilungen der positiven und negativen Beispiele in den Testmengen sind, würde man nun bei einer Dominanz von positiven Beispielen den zweiten (da er auf diesen wahrscheinlich eine höhere Genauigkeit erzielt) und bei einer höheren Anzahl von negativen Instanzen den ersten Klassifikator bevorzugen. An diesem Beispiel kann man erkennen, dass eine Bewertung über die Genauigkeit keinen Sinn macht, wenn sich die Verteilung der Beispiele in der Testmenge von der in der Testmenge unterscheidet.

In der Grafik 2.2 ist für eine angenommene Gleichverteilung der positiven und negativen Beispiele (repräsentiert durch die Linie mit der Steigung eins) der Klassifikator RIPPER am besten. Nun bildet man den Schnittpunkt der Linie, die die Verteilung repräsentiert, mit der Diagonalen, die die Punkte $(1, 0)$ und $(0, 1)$ verbindet. Hat man diesen Schnittpunkt mit der Achse verbunden, auf der die *TPR* aufgetragen ist, kann man in etwa die zugehörige Genauigkeit ablesen, die für das Beispiel bei 84 % liegt. Alle Klassifikatoren, die auf der gleichen Verteilungslinie liegen, erreichen die gleiche Genauigkeit. Es ist immer möglich, eine Verteilung in Form einer Linie im ROC-Raum vorzugeben und diese so lange zu verschieben, bis einer der Klassifikatoren auf ihr liegt. Die Genauigkeit kann dann approximiert werden, indem man für diese Verteilungslinie einen Punkt auf der *TPR-Achse* bestimmt. Dieser Punkt repräsentiert die Klassifikatoren, die eine unterschiedliche *TPR* und *FPR* besitzen, aber trotzdem die gleiche Genauigkeit erreichen (siehe Tabelle 2.3). Diese Evaluationsmethode leidet nun nicht mehr an den Problemen, die bei der Genauigkeitsbewertung aufgetreten sind.

2.5.3 Die Analyse von Heuristiken

Eine beliebige Heuristik, die als Eingabe Werte aus der Konfusionsmatrix erhält, ordnet einem Punkt im ROC-Raum einen Evaluationswert m zu. Legt man m fest, so ergeben sich unterschiedliche Kombinationen der Werte der Parameter, die das gleiche Ergebnis liefern. Anders ausgedrückt gibt es verschiedene Regeln, die denselben Evaluationswert zugeordnet

bekommen. Wählt man beispielsweise als Heuristik die Formel $wra = TPR - FPR$, setzt $m{=}0.5$ und geht von insgesamt 20 positiven und 10 negativen Beispielen aus, so genügen der Gleichung die Kombinationen aus Tabelle 2.3.

TPR	FPR
$\frac{10}{20} = 0.5$	0
$\frac{12}{20} = 0.6$	$\frac{1}{10} = 0.1$
$\frac{14}{20} = 0.7$	$\frac{2}{10} = 0.2$
$\frac{16}{20} = 0.8$	$\frac{3}{10} = 0.3$
$\frac{18}{20} = 0.9$	$\frac{4}{10} = 0.4$
$\frac{20}{20} = 1$	$\frac{5}{10} = 0.5$

Abbildung 2.3: Beispiel für verschiedene Kombinationen

Daraus ergeben sich für diesen Fall die Punkte (x, y) mit $x \in FPR$, $y \in TPR$ und einem zusätzlichen Evaluationswert der Regel R mit $h(R) \in TPR - FPR$. Verbindet man nun im ROC-Raum die Punkte, die von der Heuristik denselben Wert zugeordnet bekommen, so erhält man eine Linie. Diese einzelne Linie entspricht einer gleichen Evaluation bei unterschiedlichen Werten von FPR und TPR. Die unterschiedlichen Linien repräsentieren unterschiedliche Evaluationswerte[8] und sind eindeutig geordnet. Meistens steigt der Evaluationswert mit einer Annäherung der Linien zu dem Punkt, der die perfekte Theorie repräsentiert oder er ist beispielsweise umso größer je höher die Linien im ROC-Raum angeordnet sind.

Definition 2.11 (Isometriken) Eine Isometrik ist eine Linie in einem 2-dimensionalen Raum, die die Punkte (n_i, p_i) verbindet, die verschiedenen Regeln R_i den gleichen Heuristikwert $h(R_i)$ zuordnen, aber unterschiedlich viele positive und negative Beispiele abdecken. Alle repräsentativen Linien auf einmal bilden die Isometriken einer Heuristik.

Die Anordnung der Linien entspricht einer Verteilung der Beispiele. In der Grafik 2.4 aus [6] sind zwei verschiedene Verteilungen zu erkennen.

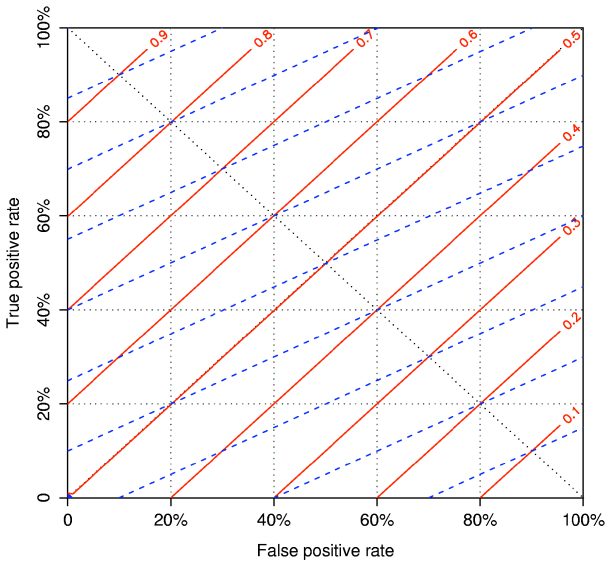

Abbildung 2.4: Isometriken von Accuracy im ROC-Raum

[8] Die Evaluationswerte decken meistens den gesamten Wertebereich der Heuristik z.B. in 0.1-er Schritten ab.

Die soliden Linien in der Grafik entsprechen einer Gleichverteilung von positiven und negativen Beispielen. Die gestrichelten Linien entstehen, wenn man von doppelt so vielen positiven Beispiele ausgeht. Es sind also zwei verschiedene Klassenverteilungen in der Abbildung 2.4 zu sehen. Es werden die Raten der richtig und falsch klassifizierten Beispiele aufgetragen und es wird keinerlei Aussage über die Gesamtanzahl von Beispielen und deren Verteilung getroffen, da die beiden Achsen normiert sind. Es ändert sich aus diesem Grund nur die Länge der Achsenabschnitte für ein Beispiel. Diese beträgt auf der *TPR-Achse* $\frac{1}{P}$ und auf der *FPR-Achse* $\frac{1}{N}$. Daher ist man gezwungen immer Linien für mehrere unterschiedliche Verteilungen zu zeichnen, da man sonst einen falschen Eindruck von der Heuristik bekommen könnte. Lässt man in der Grafik beispielsweise die gestrichelten Linien weg, dann entsprechen die Isometriken denen der oben erwähnten Heuristik *wra*[9]. Gerade für eine eindeutige Darstellung eignet sich aus diesem Grund ein Raum, in dem die Gesamtanzahl von positiven und negativen Beispielen für eine unterschiedliche Klassenverteilung aufgetragen ist[10]. Dieser wird als PN-Raum bezeichnet und im folgenden Kapitel eingehend behandelt.

[9] Bei einer Gleichverteilung von positiven und negativen Beispielen in der Trainingsmenge sind die Heuristiken *Accuracy* und *Weighted Relative Accuracy* äquivalent. (siehe (3.11))

[10] Bei einer uniformen Klassenverteilung würde man die gleichen Probleme wie beim ROC-Raum haben.

Kapitel 3

Überblick über die verwendeten Heuristiken

In diesem Kapitel wird ein Überblick über verschiedene Heuristiken gegeben. Zu Anfang werden die Isometriken der Heuristiken näher erläutert. Es folgt eine Zusammenfassung der benutzten Konzepte, die sich in die Standard-Heuristiken, die Basis-Heuristiken und die parametrisierbaren Heuristiken gliedert. Die Standard-Heuristiken sind allgemein gut bekannt und werden für einen Vergleich verwendet, um die Güte der anderen Heuristiken besser abschätzen zu können. Die vier Basis-Heuristiken beschreiben den jeweiligen Trade-Off für die parametrisierbaren Heuristiken, wobei die Heuristik *Precision* bei allen als untere Grenze vorkommt. Die Heuristiken werden einerseits mit der zugrunde liegenden Formel beschrieben und andererseits mit ihren Isometriken, da diese einen intuitiven Zugang zu der Funktionsweise ermöglichen.

3.1 Isometriken im PN-Raum

Im Unterschied zu ROC-Räumen, die im vorherigen Kapitel beschrieben wurden, beziehen sich die Isometriken der einzelnen Heuristiken auf den PN-Raum. Bei ROC-Räumen wird die TPR $\frac{p}{P}$ über der FPR $\frac{n}{N}$ aufgetragen. Beim PN-Raum, der häufig auch als „Coverage Space“ [11] bezeichnet wird, ist die Gesamtanzahl der positiven Beispiele der Trainingsmenge auf der y-Achse und die der negativen auf der x-Achse aufgetragen. Der Hauptunterschied der beiden Darstellungsmöglichkeiten besteht darin, dass im PN-Raum die absolute Anzahl von positiven und negativen Beispielen verwendet wird, während bei ROC-Kurven die Raten der richtig und falsch klassifizierten Beispiele aufgetragen werden. So muss beim ROC-Raum explizit auf die Verteilung der Beispiele eingegangen werden, während sich

diese beim PN-Raum von selbst ergibt. Jeder ROC-Raum ist in einen PN-Raum transformierbar und umgekehrt. Die Unterschiede der beiden Darstellungsmöglichkeiten sind in folgender Tabelle aus [11] aufgetragen.

Eigenschaft	ROC-Raum	PN-Raum
x-Achse	FPR $(\frac{n}{N})$	n = *die abgedeckten negativen Beispiele*
y-Achse	TPR $(\frac{p}{P})$	p = *die abgedeckten positiven Beispiele*
Koordinaten der leeren Theorie	$(0,0)$	$(0,0)$
Koordinaten der allgemeinsten Theorie	$(1,1)$	(N,P)
Auflösung	$(\frac{1}{N}, \frac{1}{P})$	$(1,1)$
Steigung der Diagonalen	1	$\frac{P}{N}$
Steigung der Linie p=n	$\frac{N}{P}$	1

Tabelle 3.1: Transformation

Im PN-Raum repräsentiert ein Punkt (n_i, p_i) eine Regel R_i, die p_i positive aus insgesamt P positiven Beispielen und n_i negative aus insgesamt N negativen Beispielen der Trainingsmenge abdeckt. Hat man eine Theorie gefunden, so werden alle Regeln im PN-Raum aufgetragen. Diese verschiedenen Punkte werden durch Linien miteinander verbunden, so dass jede Linie genau den Regeln entspricht, die von der Heuristik einen gleichen Wert zugeordnet bekommen haben und unterschiedlich viele positive und negative Beispiele der Trainingsmenge abdecken. Genau wie beim ROC-Raum handelt es sich auch beim PN-Raum um einen 2-dimensionalen Raum, der den Linien einen Wert zuordnet (die Werte, die in Grafik 3.1 an den Linien stehen). Man könnte diese Wertzuordnung auch als eine weitere Dimension betrachten. Da die Evaluationswerte für das Verständnis der Isometriken einer Heuristik aber nur bedingt nötig sind und 3-dimensionale PN-Räume schnell unübersichtlich werden, werden die Linien im Folgenden nicht beschriftet und die PN-Räume zweidimensional dargestellt.

In dem PN-Raum aus 3.1 gibt es 60 negative und 48 positive Beispiele. Diese Verteilung wird für alle folgenden PN-Räume angenommen. Der untere Punkt entspricht der Regel *R1*, die 30 negative und 20 positive Beispiele abdeckt und einen Evaluationswert von $h(R1) = 0.4$ erhält. Der zweite Punkt repräsentiert die Regel *R2,* die 48 negative und 31 positive Beispiele abdeckt und gleich evaluiert wird. Die n-Achse entspricht bei dieser Heuristik einem Evaluationswert von 0.0 und die p-Achse einem Wert von 1.0[1].

Betrachtet man den PN-Raum im Kontext des Lernprozesses und nicht zur Visualisierung von Heuristiken, so kann man sich das Lernen von Regeln als einen Pfad durch den PN-Raum vorstellen. Während des Lernens kommt in einen PN-Raum, in dem sich bereits der

[1] Dem vorgebildeten Leser wird aufgefallen sein, dass es sich bei der skizzierten Heuristik um *Precision* handelt. Da die Verteilung der positiven und negativen Beispiele nicht gleich ist, erhält die Diagonale nicht den Evaluationswert 0.5, wie es bei einer Gleichverteilung der Beispiele der Fall wäre.

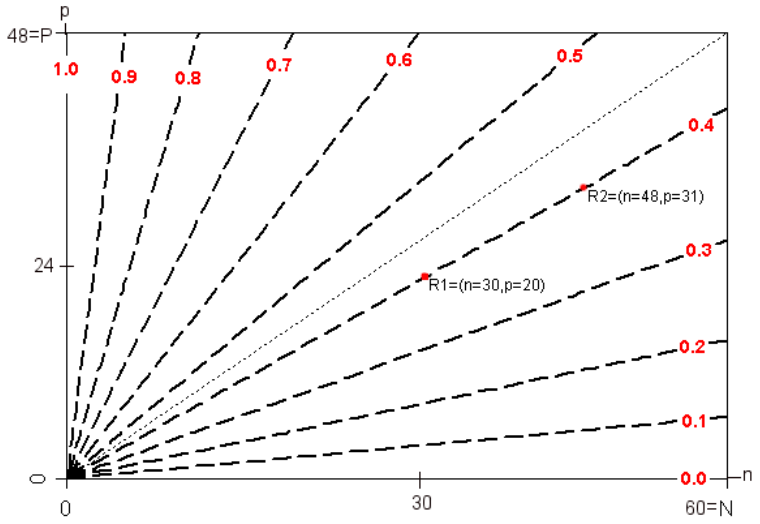

Abbildung 3.1: Beispiel zum PN-Raum

Punkt (n_1, p_1) befindet, der Punkt (n_2, p_2) hinzu, wenn der Theorie eine neue Regel R_2 hinzugefügt wird. Üblicherweise wird sich ein neuer Punkt in Richtung (N, P) orientieren, da mit jeder neuen Regel eine höhere Abdeckung auf der Trainingsmenge erreicht wird. Wenn mit einer Regelmenge alle Beispiele der Trainingsmenge abgedeckt sind, befindet sich der Punkt (n_i, p_i) exakt beim Punkt (N, P). Dieser entspricht der universellen Theorie. Sie besteht aus einer einzigen Regel, die mit der Bedingung *True* repräsentiert wird. Diese generellste Regel wird durch folgendes Konstrukt gebildet:

$$(TRUE) \rightarrow positiv \tag{3.1}$$

Üblicherweise wird beim Regel-Lernen aber versucht sich dem Punkt $(0, P)$ immer weiter anzunähern, da dieser genau die Theorie repräsentiert, die alle positiven Beispiele, aber kein einziges negatives Beispiel abdeckt. Erreicht man diesen Punkt, so hat man die optimale (oder auch perfekte) Theorie gefunden. Häufig ist aber eher eine Annäherung zu dem Punkt (N, P) zu erkennen, da durch das Hinzufügen von neuen Regeln auch negative Beispiele mit abgedeckt werden. Ist ein Abbruchkriterium vorgegeben, nach dem man alle positiven Beispiele abdecken muss, so landet man durch das Hinzufügen der letzten Regel zwangsläufig an der Stelle $(x, P).x \in [0, N]$ (siehe Definition 5.1). Man kann sich das Lernen einer Theorie als Weg durch den PN-Raum visualisieren. Begonnen wird am Punkt $(0, 0)$, an dem kein einziges Beispiel abgedeckt wird. Fügt man der Theorie weitere Regeln hinzu, so wird man sich schrittweise dem Punkt (N, P) annähern, während man mit jeder Regel in einen neuen Unterraum des PN-Raumes gelangt.

Man kann auch das Generalisieren oder Spezialisieren einer einzelnen Regel im PN-Raum visualisieren. Es wird entweder analog wie bei dem Lernen einer Theorie beim Punkt $(0, 0)$ begonnen oder bei (N, P). Entfernt man nun Bedingungen aus dem Regelkörper, so nähert

man sich dem Punkt (N, P) (Bottom-Up) an. Fügt man Bedingungen hinzu, so bewegt man sich in Richtung $(0, 0)$ (Top-Down). Ein einzelner Punkt entspricht einer Kandidatenregel, die eine gewisse Anzahl an Bedingungen hat.

Betrachtet man sich die Isometriken der Heuristiken, so unterscheiden sie sich in der Anordnung der Linien im PN-Raum. Handelt es sich um gebogene Linien, so spricht man von nichtlinearen Heuristiken. Die linearen Heuristiken gliedern sich in solche, deren Linien parallel sind und die, deren Linien um den Ursprung des PN-Raums rotieren. Die parallelen Linien entsprechen analog wie beim ROC-Raum einer bestimmten Klassenverteilung. Die Verteilung der positiven und negativen Beispiele ist im PN-Raum konstant. Sobald aber eine Regel gefunden wurde und die abgedeckten Beispiele aus der Trainingsmenge entfernt worden sind, ändert sich auch in diesem Raum die Verteilung. Im Folgenden kann daher für diese Darstellungsform angenommen werden, dass die Klassenverteilung und die Kosten einer fehlerhaften Klassifikation äquivalent sind. Der erste Ansatz geht von bekannten oder angenommenen Kosten aus, während das Modell der rotierenden Linien auf unbekannten Kosten beruht. Hier wird in jedem einzelnen Schritt die Linie mit der steilsten Steigung ausgewählt, woraus folgt, dass die Kosten ständig variieren. Die Verteilung von positiven und negativen Beispielen in der Testmenge ist also im ersten Fall, unabhängig vom Rückgabewert der Heuristik, irrelevant (siehe Abbildung 3.2). Im zweiten Fall variiert die Verteilung von positiven und negativen Beispielen, je nachdem welcher Evaluationswert der Regel zugeordnet wurde[2] (siehe Abbildung 3.3). Im nichtlinearen Fall ändert sich die Verteilung sogar für ein und dieselbe Linie, da diese gebogen ist. Der nichtlineare Fall wird bei der Heuristik *Correlation* nur am Rande behandelt und lediglich bei einigen Parametrisierungen des *Klösgen-Maßes* intensiver betrachtet. Das Hauptaugenmerk liegt aber auf den linearen Heuristiken, deren unterschiedliche Basismodelle in [9] als h_{costs} und h_{pr} bezeichnet werden. Das Modell der parallelen Linien lässt sich durch die generelle Form

$$h_{costs} = a * p - b * n \sim c * p - (1 - c) * n \sim p - d * n \tag{3.2}$$

darstellen. Bei dieser Kostenmetrik ist es gleich gut ein positives Beispiel abzudecken oder $\frac{c}{1-c}$ negative Beispiele auszuschließen. Je nachdem wie man die Variable c setzt, ergeben sich verschiedene Heuristiken, deren Isometriken immer aus parallelen Linien bestehen. Die Heuristik *Accuracy* nimmt beispielsweise die gleichen Kosten für die Abdeckung eines positiven Beispiel und eines negativen an (mit $c = \frac{1}{2}$). Diese angenommenen Kosten verändern sich während des Lernprozesses nicht. Ist es nun möglich die Variablen a und b oder c oder d des generellen Kostenmodells so zu setzen, dass sich daraus die Formel der Heuristik ergibt, so hat man nachgewiesen, dass diese Heuristik ein Repräsentant des bestimmten Modells ist.

Beim $h_{pr} - Modell$ sind die Linien der Isometrik nicht mehr parallel, sondern rotieren um

[2] Beziehungsweise welche Linie als beste ausgewählt wurde.

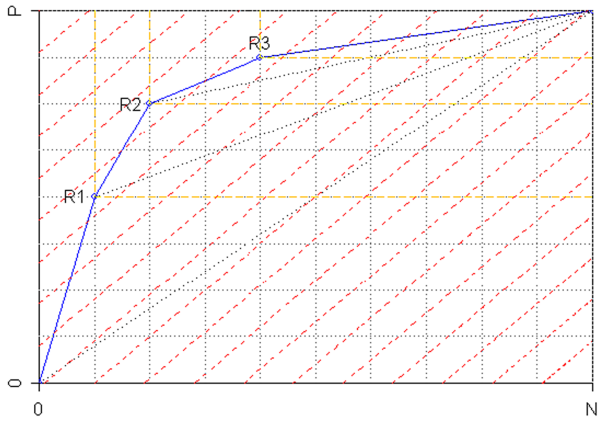

Abbildung 3.2: Isometriken für *Accuracy* in verschachtelten PN-Räumen

den Ursprung. Alle Heuristiken, deren Linien diese Eigenschaft aufweisen, sind Vertreter des $h_{pr} - Modells$.

Die beiden Modelle unterscheiden sich in einigen interessanten Punkten, auf die nun genauer eingegangen wird. Eine wichtige Eigenschaft des $h_{costs} - Modells$ ist, dass ein lokales Optimum im PN-Raum auch ein globales Optimum ist, da verschachtelte PN-Räume bei parallelen Linien deren Winkel nicht verändern. In der Grafik 3.2 aus [10] repräsentiert der Punkt R1 den Anfangszustand des Regel-Lerners nachdem die von der ersten gefundenen Regel abgedeckten Beispiele entfernt worden sind. Es verbleiben nun genau die Beispiele in der Trainingsmenge, auf die bisher noch keine Regel zutrifft. Man befindet sich daher nach dem ersten Schritt in einem Unterraum des vorherigen Raumes. Sobald der Theorie eine neue Regel hinzugefügt wird, werden alle Beispiele entfernt, die von dieser abgedeckt werden. Daher kommt man schrittweise in kleinere Unterräume. Trägt man diesen Sachverhalt in einem einzigen PN-Raum auf, so spricht man von verschachtelten PN-Räumen. Betrachtet man nun die Isometriken einer Heuristik des $h_{costs} - Modells$, in diesem Beispiel *Accuracy*, so kann man erkennen, dass die Winkel der Linien durch den Wechsel in einen neuen Unterraum nicht verändert werden und daher die Kosten überall gleich bleiben.

Des weiteren leidet dieses Modell seltener an *Overfitting*, da man beispielsweise bei der Heuristik Accuracy $h_{acc} = p - n$ leicht eine generelle Regel finden kann, deren Heuristikwert größer als eins ist. Da diese Regel gut bewertet wird, aber trotzdem noch viele Beispiele abdeckt, resultiert eine allgemeinere Theorie, die nicht übermäßig genau an die Trainingsmenge angepasst ist.

Im Gegensatz dazu ist das $h_{pr} - Modell$ anfälliger für *Overfitting*, da hier der maximale Heuristikwert eins ist. Aus diesem Grund wird beispielsweise von der Heuristik Precision $h_{pr} = \frac{p}{p+n}$ eine Regel, die genau ein positives und kein negatives Beispiel abdeckt, mit dem höchsten Heuristikwert belohnt. Deckt man mit der Theorie jedes positive Beispiel mit einer

eigenen Regel ab, so wird der Klassifikator exakt an die Trainingsmenge angepasst sein, aber ungesehene Beispiele schlecht klassifizieren. Außerdem ist ein lokales Optimum in diesem Modell nicht zwingend global optimal, da sich der Ursprung in einen anderen Unterraum verschiebt und die Linien dann in diesem Unterraum um den aktuellen Ursprung rotieren. Die Winkel der Linien werden dadurch verändert, was daran liegt, dass die Unterräume in diesem Modell invariant sind. Dieses Verhalten wird in der Grafik 3.3 aus [10] zu den Isometriken von *Precision* in verschachtelten PN-Räumen deutlich.

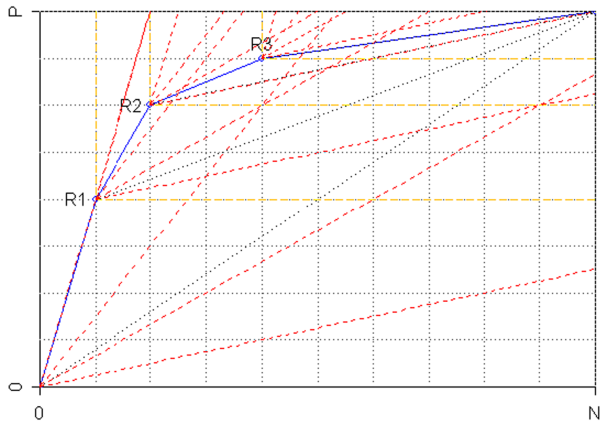

Abbildung 3.3: Isometriken für *Precision* in verschachtelten PN-Räumen

Beide Modelle haben ihre Vor- und Nachteile. Es ist noch nicht klar, ob die beschriebenen Ansätze die bestmöglichen sind. Es könnte sein, dass ein nichtlineares Maß viel besser abschneidet, als alle Vertreter der beiden erläuterten Basismodelle. Bei den parametrisierbaren Heuristiken hat das *Klösgen-Maß* häufig nichtlineare Isometriken und wird aus diesem Grund besonders ausführlich behandelt. Alle anderen Heuristiken bis auf *Correlation* weisen lineare Isometriken auf. In diesem Abschnitt ist es aber vor allem wichtig gewesen, die Isometriken näher zu erläutern, da sie für eine gute Visualisierung der verschiedenen Heuristiken unabdingbar sind. Beim Betrachten der Isometriken einer Heuristik bekommt man häufig einen intuitiven Zugang zur Funktionsweise als über deren Formel.

3.2 Standard-Heuristiken

Es folgt nun ein Überblick über alle benutzten Heuristiken. Anfangs wird die Funktionsweise einer Heuristik und deren Ziel erläutert. Danach wird auf die Standard-Heuristiken näher eingegangen. Diese werden für einen Vergleich mit den parametrisierbaren Heuristiken benötigt. Die Heuristiken sind durch Evaluationsfunktionen beschrieben, die die Variablen aus der Tabelle 2.2 aus Abschnitt 2.3 als Parameter verwenden. An diesen Funktionen lässt sich erkennen, welche Strategie den verschiedenen Heuristiken zugrunde liegt. Wie

in Abschnitt 2.3 aufgezeigt worden ist, sind Heuristiken das Kernstück eines Regel-Lern Algorithmus. Aus diesem Grund wird in diesem Abschnitt ausführlich auf die Konzepte eingegangen, die der Erstellung einer guten Heuristik zu Grunde liegen.

In Kapitel 2 ist deutlich geworden, dass Regel-Lerner intern immer nur mit 2-Klassen-Problemen umgehen können. Aus diesem Grund kann man sich die Vorhersagen des Klassifikators anhand der Konfusionsmatrix aus Tabelle 2.3 vorstellen. Man unterscheidet zwischen positiv und negativ vorhergesagten Beispielen. Alle Beispiele, die von der Regelmenge abgedeckt werden, entsprechen einer positiven Vorhersage und alle die nicht abgedeckt werden einer negativen Klassifikation. Die Maße errechnen die Güte anhand der Qualität der Konfusionsmatrix. Doch welche Ziele sollte eine gute Heuristik erreichen? Dazu soll im Folgenden versucht werden, die Konzepte, auf denen eine sinnvollen Heuristik beruht, schrittweise zu erläutern.

Generell versucht man beim Regel-Lernen immer alle positiven Beispiele (erreiche Vollständigkeit) und kein negatives Beispiel (erreiche Konsistenz) abzudecken. Daher sollte eine optimale Heuristik beide Ziele simultan erreichen.

- MaximizePositive

$$h_{max\_p} = p \tag{3.3}$$

Bei diesem Konzept wird versucht so viele positive Beispiele wie möglich abzudecken, unabhängig davon wie viele negative Beispiele abgedeckt werden. Hat man zum Beispiel zwei Regeln R_1 und R_2 wobei R_1 1 positives Beispiel und 1000 negative Beispiele abdeckt und R_2 ein positives und kein negatives Beispiel, so erhalten beide Regeln denselben Heuristikwert. Daher gilt $h_{max\_p}(R_1) = h_{max\_p}(R_2)$. Dieses Prinzip ist äquivalent zum *Recall* aus dem Information Retrieval oder zum *Support* bei Assoziationsregeln.

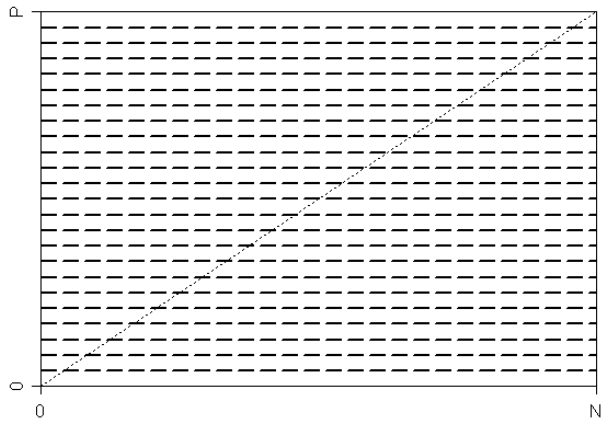

Abbildung 3.4: Isometriken für *MaximizePositive*

Mit dem *Recall* wird genau die Vollständigkeit der Theorie beschrieben. Je mehr positive Beispiele von allen positiven Beispielen abgedeckt sind, desto höher ist der Wert. Auf dieses Konzept wird in Kapitel 3.3.2 näher eingegangen, da es hier als Heuristik zur Abdeckungsoptimierung verwendet wird. *MaximizePositive* ist das erste Ziel, das eine gute Heuristik erreichen sollte.

Bei den Isometriken handelt aus Abbildung 3.4 es sich um parallele Linien, die horizontal liegen. Diese Heuristik versucht den Regel-Lerner in Richtung der Linie $(x, P).x \in [0, N]$ zu steuern, da auf dieser Linie alle positiven Beispiele abgedeckt werden. Je höher die Linie liegt, desto größer ist der zugeordnete Wert. Diese Heuristik ist im $h_{costs} - Modell$ angesiedelt, was an der Variablensetzung $a = 1$ und $b = 0$ zu erkennen ist, die sich für diese Heuristik ergibt:

$$h_{costs} = a * p - b * n = 1 * p - 0 * n = p = h_{max\_p} \quad (3.4)$$

- MinimizeNegative

$$h_{min\_n} = -n \quad (3.5)$$

Bei dem Konzept *MinimizeNegative* wird versucht so wenig negative Beispiele wie möglich abzudecken, ohne darauf zu achten, wie viele positive Beispiele abgedeckt werden. Hierbei handelt es sich um das zweite der beiden Ziele, die eine gute Heuristik erreichen sollte.

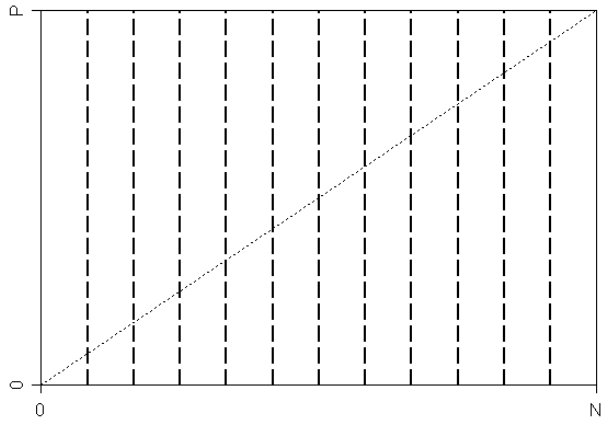

Abbildung 3.5: Isometriken für *MinimizeNegative*

In den Isometriken ist zu erkennen, dass es sich um parallele Linien handelt. Da die Anzahl der abgedeckten positiven Beispiele unerheblich ist, bilden sich vertikale Linien. Die beste Bewertung erhält die Linie, die sich am weitesten links befindet, da hier am wenigsten negative Beispiele abgedeckt werden. Diese Heuristik ist ebenfalls ein Vertreter des $h_{costs} - Modells$. Dies wird deutlich, wenn man die Variablen des generellen Modells auf $a = 0$ und $b = 1$ setzt.

Jede Heuristik versucht diese beiden Konzepte möglichst simultan zu optimieren. Da das Ziel eines guten Regel-Lern-Algorithmus darin besteht, sich mit der Regelmenge dem Punkt $(0, P)$ anzunähern oder diesen möglichst zu erreichen, muss man die beiden obigen Konzepte gleichzeitig erreichen. Hat man das perfekt geschafft, deckt also alle positiven und kein negatives Beispiel ab, so ist die optimale Theorie gefunden worden.

3.2.1 Accuracy

$$h_{acc} = p - n \cong \frac{p+(N-n)}{P+N} \tag{3.6}$$

Bei dieser Heuristik wird die Genauigkeit einer Regel evaluiert, indem die Prozentzahl der richtig klassifizierten Beispiele gemessen wird. Da P und N konstant sind für alle Kandidatenregeln, genügt es, sich für die Bewertung der Regel auf die aktuell abgedeckten Beispiele zu beziehen. *Accuracy* stellt eine naive Kombination der beiden Heuristiken *MaximizePositive* und *MinimizeNegative* dar. Es werden beide Konzepte addiert, da so die Ziele dieser elementaren Ansätze gleichzeitig zum Tragen kommen, was man auch in den Isometriken erkennen kann. Sie bestehen aus Linien im 45°-Winkel. Diese 45°-Steigung entspricht der Abdeckung von gleich vielen positiven und negativen Beispielen, unabhängig davon, wie die Gesamtverteilung der Beispiele ist. Ein Problem dieser Heuristik besteht deshalb darin, dass das Abdecken eines einzigen positiven Beispiels oder das Ausschließen eines einzigen negativen Beispiels von der Heuristik gleich gut bewertet wird.

Dazu ein kurzes Beispiel:

Mit *Accuracy* ist eine Regel R_1 bewertet worden, die 50 positive und 20 negative Beispiele abdeckt. Der Heuristikwert ist $h_{acc}(R_1) = p - n = 50 - 20 = 30$

Nun führt eine weitere Generalisierung/Verfeinerung von R_1 zur Regel R_2.

1. Fall (Generalisierung): R_2 deckt nun 10 positive Beispiele mehr ab als R_1, also 60 positive und 20 negative Beispiele. Der Heuristikwert ist daher $h_{acc}(R_2) = 60 - 20 = 40$

2. Fall (Spezialisierung): R_2 deckt nun 10 negative Beispiele weniger ab als R_1, also 50 positive und 10 negative Beispiele. Der Heuristikwert ist ebenfalls $h_{acc}(R_2) = 50 - 10 = 40$

Hierbei ist die Verteilung der Beispiele in der Testmenge völlig unerheblich. Gibt es beispielsweise 1000 positive und lediglich 20 negative Beispiele, so wäre selbstverständlich R_2 eine bessere Regel als R_1, da sie 10 negative Beispiel weniger abdeckt. Da sich die

Bewertung einer Regel mit der Heuristik *Accuracy* immer nur auf die aktuell abgedeckten Beispiele bezieht und die Verteilung der Beispiele außer Acht lässt, gilt überall im PN-Raum die gleiche Bewertung.

Diese Eigenschaft muss nicht zwingend ein Problem darstellen. Sobald aber die *a priori*-Verteilung der positiven und negativen Beispiele der Trainingsmenge für die Domäne nicht repräsentativ ist oder die Kosten einer fehlerhaften, beziehungsweise der Profit einer positiven Klassifizierung unbekannt sind, führt dies zu einem unerwünschten Verhalten. Es ist in dann nicht mehr gleichbedeutend ein positives Beispiel mehr oder ein negatives weniger abzudecken.

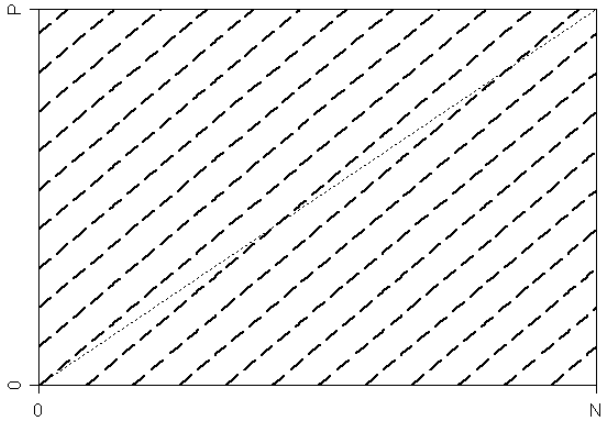

Abbildung 3.6: Isometriken für *Accuracy*

In den Isometriken von *Accuracy* wird deutlich, dass auch sie dem $h_{costs} - Modell$ entsprechen. Setzt man die Variablen der generellen Form des $h_{costs} - Modells$ wie folgt, so erhält man die Heuristik *Accuracy:* entweder $a = b = d = 1$ oder $c = \frac{1}{2}$.

3.2.2 Correlation

$$h_{corr} = \frac{p*(N-n)-n*(P-p)}{\sqrt{P*N*(p+n)*(P-p+N-n)}} \tag{3.7}$$

Die Heuristik *Correlation* misst die Korrelation zwischen der Klassenvorhersage des Regel-Lerners und der tatsächlichen Klasse. Sie entspricht einem $\chi^2 - Test$ wie in [11] gezeigt wurde. Der $\chi^2 - Test$ gibt an, mit welcher Wahrscheinlichkeit eine Verteilung signifikant ist. Es wird die obige Formel verwendet, die auf die Parameter aus der Konfusionsmatrix zugeschnitten ist. Der Wertebereich bei *Correlation* ist $[-1, 1]$. Je höher die Korrelation ist, desto besser wird die Regel bewertet. Die Idee wird auch bei anderen Problemstellungen angewandt. So wird dieselbe Heuristik als *Pre-Pruning* Strategie verwendet. Dieses Konzept wurde in Abschnitt 2.1.3 näher erläutert. Es wird ein Threshold festgelegt, ab dem eine Regel überhaupt zur Theorie hinzugefügt werden darf.

Der $\chi^2 - Test$ basiert auf der Konfusionsmatrix.

Definition 3.1 ($\chi^2 - Test$) $\chi^2 = \sum_{i,j} \frac{(k_{ij} - E(k_{ij}))^2}{E(k_{ij})} = \frac{total*(k_{11}*k_{00} - k_{10}*k_{01})^2}{(k_{11}+k_{10})*(k_{01}+k_{00})*(k_{11}+k_{01})*(k_{10}+k_{00})}$
mit

- $k_{i,j} \equiv$ die beobachteten Werte in Zeile i und Spalte j der Konfusionsmatrix und
- $E(k_{i,j}) \equiv$ die erwarteten Werte in Zeile i und Spalte j der Konfusionsmatrix

Der hintere Term bezieht sich direkt auf die Spalten und Zeilen der Matrix, funktioniert also daher nur für 2-Klassen-Probleme. Angenommen man hat folgende Matrix:

	positiv vorhergesagt	negativ vorhergesagt	
tatsächlich positiv	40	60-40=20	60
tatsächlich negativ	12	30-12=18	30
	57	33	*total=90*

Tabelle 3.2: Beispieltabelle zum $\chi^2 - Test$

Es ergibt sich dann dieser Wert:

$\chi^2 = \frac{90*(18*40-12*20)^2}{(18+12)*(20+40)*(18+20)*(12+40)} = 5.82996$

Man kann nun in einer Tabelle[3] ablesen, mit welcher Wahrscheinlichkeit diese Verteilung signifikant ist. In dem Beispiel liegt eine hohe Wahrscheinlichkeit von 0.98 vor. Benutzt man die Formel von der Heuristik, ergibt sich eine Korrelation von 0.2545. Deckt eine Regel ausschließlich positive/negative Beispiele ab und kein einziges negatives/positives Beispiel, so liegt die höchstmögliche/niedrigstmögliche Korrelation von 1 / -1 vor, die einer perfekten positiven/negativen Korrelation entspricht.

Die Heuristik *Correlation* ist an der Diagonalen symmetrisch und bewertet die Abdeckung von vielen positiven Beispielen gleich wie das Ausschließen von vielen negativen. Dieser Effekt wird durch die Biegung der Linien in den Bereichen von kleinen und großen Werten von p und n deutlich. Die Isometriken sind nichtlinear, daher ist sie kein Repräsentant von einem der beiden Modelle.

Interessant an den Isometriken von *Correlation* ist, dass die Bewertung auf der Diagonalen genau 0 ist. Alle Linien die sich rechts von der Diagonalen befinden haben negative Werte und alle die sich links befinden positive Werte. So bekommt zum Beispiel der Punkt $(N, 0)$ den Wert -1 und der Punkt $(0, P)$ die beste Bewertung von eins.

[3] Eine Tabelle der Werte χ^2 einer chi-quadrat-verteilten Zufallsvariable für vorgegebene Werte der Verteilungsfunktion $F(\chi^2)$ mit einem Freiheitsgrad.

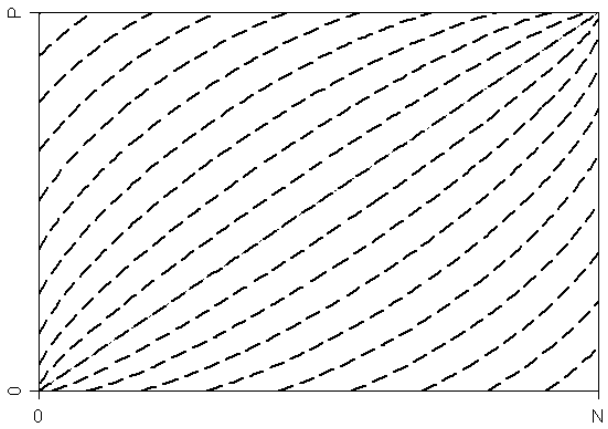

Abbildung 3.7: Isometriken für *Correlation*

3.2.3 Laplace

$$h_{lap} = \frac{p+1}{p+n+2} \tag{3.8}$$

Diese Heuristik arbeitet ähnlich wie *Precision*, nur dass nicht bei null angefangen wird, die Beispiele zu zählen. Es wird davon ausgegangen, dass jede Regel *a priori* eine gewisse Anzahl an Beispielen abdeckt. Bei *Laplace* wird ein positives sowie ein negatives Beispiel im Vorhinein abgedeckt. Dies resultiert in einer Bewegung des Ursprungs des PN-Raumes.

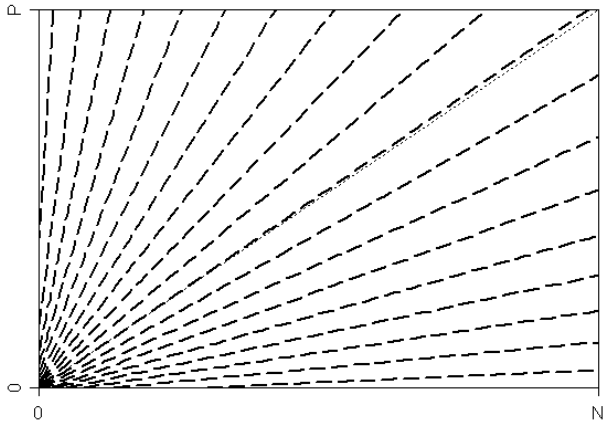

Abbildung 3.8: Isometriken für *Laplace*

Hier soll aber die Heuristik *Laplace* nicht weiter behandelt werden. Genauer wird sie in Abschnitt 3.4.2 zum *m-Estimate* beschrieben.

Die Linien in der Isometrik rotieren um den Punkt $(-1, -1)$. Daher siedelt sich die Heuristik *Laplace* im $h_{pr} - Modell$ an.

3.3 Basis-Heuristiken

Die Basis-Heuristiken bilden die jeweilige Basis für die parametrisierbaren Heuristiken, die in dieser Arbeit untersucht worden sind. Hierbei wird zwischen einer Genauigkeitsoptimierung und einer Abdeckungsoptimierung unterschieden. Zum Messen der Genauigkeit einer Regel wird bei jeder parametrisierbaren Heuristik *Precision* verwendet. Für die Abschätzung der Abdeckung einer Regel werden die verschiedenen Maße *Recall*, *Weighted Relative Accuracy* und *Coverage* benutzt. Diese Heuristiken versuchen nicht eine möglichst gute Theorie zu finden, sondern optimieren ausschließlich die Abdeckung einer Regel. So wird der Heuristikwert immer besser, je mehr Beispiele insgesamt abgedeckt werden. Die Abdeckung ist allerdings bei den drei Basis-Heuristiken unterschiedlich definiert. Bei *Coverage* ist man auf ein Einschließen von positiven und negativen Beispielen aus, während beim *Recall* nur die Abdeckung der positiven Beispiele relevant ist. *Weighted Relative Accuracy* hingegen versucht, so viele positive Beispiele wie möglich einzuschließen, aber dabei keine negativen Beispiele mit abzudecken. Daher fällt diese Basis-Heuristik aus dem Konzept der reinen Abdeckung, ohne das Ziel den Lernprozess in eine sinnvolle Richtung zu lenken, heraus.

3.3.1 Precision

$$h_{pr} = \frac{p}{p+n} \tag{3.9}$$

Die Heuristik *Precision* bestimmt die Güte einer Regel durch deren Genauigkeit. Je mehr Beispiele aus der Trainingsmenge korrekt klassifiziert werden, desto höher ist die Genauigkeit einer Regel. Es wurde in [10] gezeigt, dass die Isometrik von Precision äquivalent zu der Pruning-Heuristik $h_{RIPPER} = \frac{p-n}{p+n}$ von *RIPPER* [3] ist, auf die in Kapitel 2.4 näher eingegangen wurde. Die Heuristik *Precision* bewertet alle Regeln die nur positive Beispiele abdecken mit dem höchstmöglichen Wert eins und alle die nur negative Beispiele abdecken mit dem niedrigsten Wert null. Der beste Heuristikwert entspricht genau der P-Achse des PN-Raumes, da hier kein negatives Beispiel und beliebig viele positive Beispiele abgedeckt werden. Der schlechteste Wert ist dementsprechend die N-Achse, da hier beliebig viele negative und kein positives Beispiel abgedeckt sind (vergleiche Abbildung 3.1). Alle anderen Werte werden durch eine Rotation um den Ursprung erzeugt. Im Ursprung selber ist der Heuristikwert undefiniert. In einem anderen Bereich, dem Finden von Assoziationsregeln, wird *Precision* auch als *Confidence* bezeichnet.

Die Isometriken von *Precision* repräsentieren das $h_{pr} - Modell$. Diese Heuristik tendiert dazu, die Linie mit der steilsten Steigung zu bevorzugen, da hier der Evaluationswert am größten ist.

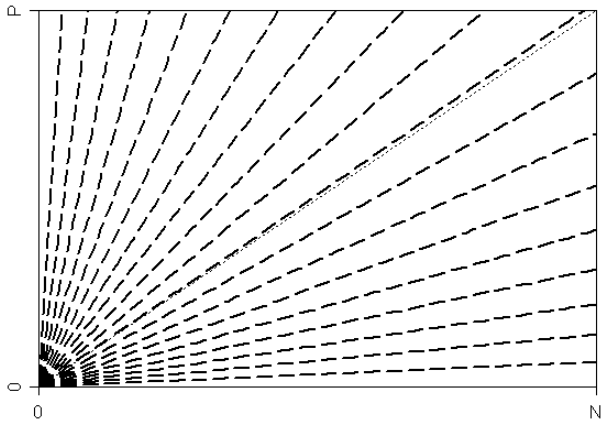

Abbildung 3.9: Isometriken für *Precision*

3.3.2 Recall

$$h_{Recall} = \frac{p}{P} \tag{3.10}$$

Das Abdeckungsmaß *Recall* ist ein äquivalentes Konzept zu *MaximizePositive.* Der einzige Unterschied besteht darin, dass noch mit der Gesamtanzahl der positiven Beispiele normiert wird. Mit diesem Maß wird errechnet, wie viele positive Beispiele von der Regel abgedeckt werden. Die Isometriken von *Recall* sind ebenfalls äquivalent zu denen von *MaximizePositives.*

3.3.3 Weighted Relative Accuracy

$$h_{wra} = \frac{p}{P} - \frac{n}{N} = TPR - FPR \tag{3.11}$$

Definition 3.2 (Weighted Relative Accuracy) $wra = \frac{p+n}{P+N} * (\frac{p}{p+n} - \frac{P}{P+N})$

Wie in [9] gezeigt wurde, gilt $h_{wra} \sim wra$. Um der Gleichgewichtung der Heuristik *Accuracy* beim Abdecken eines positiven oder Ausschließen eines negativen Beispiels entgegenzuwirken, wird bei *Weighted Relative Accuracy* mit der Gesamtanzahl der positiven und negativen Beispiele normiert. Als Ergebnis erhält man so eine Heuristik, die eine Steigerung der *TPR* und eine Verminderung der *FPR* gleich bewertet. Setzt man voraus, dass eine Gleichverteilung zwischen der Gesamtzahl der negativen und positiven Beispielen besteht, so gilt $h_{acc} = h_{wra}$. In den Isometriken wird dieser Effekt deutlich.

Das Ziel dieser Heuristik ist die Steuerung des Lernalgorithmus zum Punkt $(0, P)$. Die Linie die am nähesten an diesem Punkt ist, bekommt die höchste Bewertung. Daher zielt *Weighted Relative Accuracy* nicht auf eine reine Abdeckung ab, wie die anderen Basis-Heuristiken.

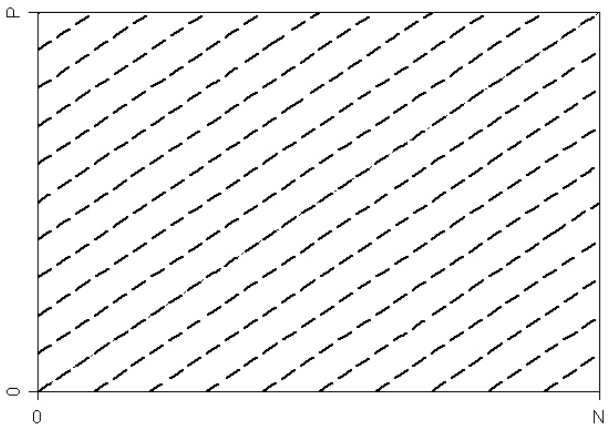

Abbildung 3.10: Isometriken für *Weighted Relative Accuracy*

In den Isometriken wird diese Präferenz an den Punkten $(0,0)$ und (N,P) deutlich, da sie den gleichen Evaluationswert erhalten, aber eine signifikant unterschiedliche Abdeckung repräsentieren.

Man erkennt nun, dass die Linien parallel zur Diagonalen $\frac{P}{N}$ sind. Aus diesem Grund ist *Weighted Relative Accuracy* genau wie *Accuracy* im $h_{costs} - Modell$ anzusiedeln. Dies kann man durch folgende Setzung der Variablen des generellen $h_{costs} - Modells$ zeigen: $a = \frac{1}{P}$ und $b = \frac{1}{N}$ oder $c = \frac{N}{P+N}$ oder $d = \frac{P}{N}$.

3.3.4 Coverage

$$h_{coverage} = \frac{p+n}{P+N} \tag{3.12}$$

Das Abdeckungsmaß Coverage dient der Bewertung der Abdeckung einer Regel. Ein höherer Wert ergibt sich, je mehr Beispiele insgesamt von der Regel abgedeckt werden. Als gute Heuristik ist dieser Ansatz selbstverständlich ungeeignet, da auch hier nur die gesamte Abdeckung bewertet wird und keine Unterscheidung zwischen positiven und negativen Beispielen gemacht wird. Üblicherweise wird die Genauigkeit eines Algorithmus, der mit dieser Heuristik arbeitet, sehr gering sein. Da Coverage hier aber genutzt wird um eine Konvergenz bei einer parametrisierbaren Heuristik zu beschreiben, ist eine geeignete Bewertung der Abdeckung für dieses Konzept wichtig.

Es wird versucht den Regel-Lerner in Richtung des Punktes (P,N) zu lenken, da dieser der höchstmöglichen Abdeckung entspricht. Respektive dazu bekommt die Linie, die sich am nähesten an diesem Punkt befindet, den höchsten Wert zugeordnet. Auch diese Heuristik ist ein Repräsentant des $h_{costs} - Modells$, zum Beispiel mit $a = \frac{1}{P+N}$ und $b = -\frac{1}{P+N}$.

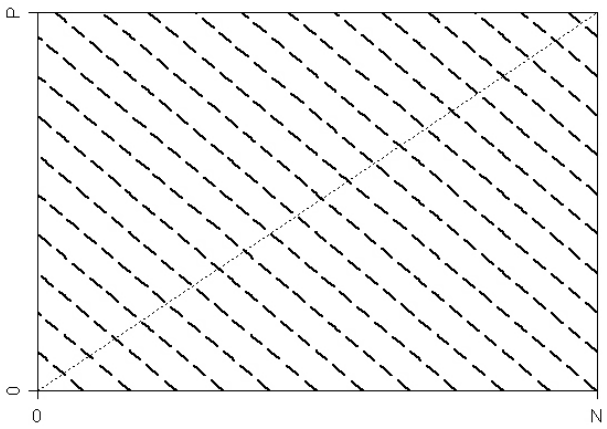

Abbildung 3.11: Isometriken für *Coverage*

3.4 Die parametrisierbaren Heuristiken

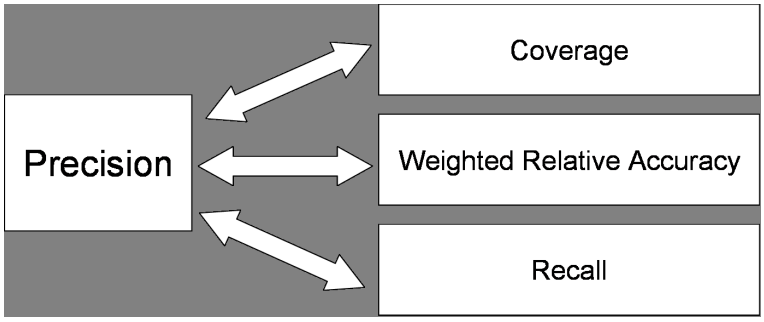

Abbildung 3.12: Trade-Offs

Definition 3.3 (parametrisierbare Heuristik) Eine parametrisierbare Heuristik verfügt über einen zusätzlichen Parameter m. Sie wägt zwischen zwei Basis-Heuristiken h_{B1} und h_{B2} ab. Mit dem Parameter kann die Tendenz zur jeweiligen Basis-Heuristik gesteuert werden, wobei gilt:

- $m \to 0$: Tendenz zur Heuristik h_{B1}
- $m \to \infty$: Tendenz zur Heuristik h_{B2}

Die parametrisierbaren Heuristiken bilden Trade-Offs zwischen den Basis-Heuristiken wie in Abbildung 3.12 zu sehen ist. Mit dem Parameter kann man steuern in welche Richtung die Heuristik tendiert. Wählt man sehr niedrige Werte so nähern sich alle parametrisierbaren Heuristiken *Precision* an, wo versucht wird die Genauigkeit einer Regel unabhängig von ihrer Abdeckung zu optimieren. Je größer man den Parameter wählt, desto mehr tendieren

die Heuristiken zu Regeln mit einer höheren Abdeckung. Die Bewertung der Abdeckung ist bei den drei parametrisierbaren Heuristiken unterschiedlich. Beim *Klösgen-Maß* wird zur Evaluation die Gesamtmenge von abgedeckten Beispielen verwendet. Das *F-Measure* benutzt nur die abgedeckten positiven Beispiele. Da der Regel-Lerner möglichst wenig negative Beispiele abdecken sollte, ist dieses Konzept ähnlich wie beim *Klösgen-Maß*. Die Genauigkeiten der beiden Abdeckungsmaße sind in etwa gleich. Beim *m-Estimate* wird von der positiven Abdeckung die negative abgezogen. Dies entspricht einer guten Heuristik und die Genauigkeit von diesem Konzept ist deutlich größer als bei den beiden anderen. Der Trade-Off des *m-Estimates* entspricht einer Abwägung zwischen den beiden Basismodellen h_{pr} und h_{costs}, da für kleine Parameter zu *Precision*, mit um den Ursprung rotierenden Linien, tendiert wird und für große Parametrisierungen zu der Basis-Heuristik *Weighted Relative Accuracy*, welche parallele Linien ausbildet. Um einen guten Trade-Off zwischen der Genauigkeit und der Abdeckung zu erreichen, sollte der Parameter die Tendenzen zu den beiden Basis-Heuristiken so justieren, dass Regeln gefunden werden, die möglichst viele Beispiele abdecken aber dabei noch eine hohe Genauigkeit erlangen.

3.4.1 Das Klösgen-Maß

Das *Klösgen-Maß* [16] bildet einen Trade-Off zwischen *Precision Gain* und dem Abdeckungsmaß *Coverage* aus Abschnitt 3.11, welches in diesem Abschnitt mit c bezeichnet wird. *Precision Gain* beschreibt die Differenz zwischen dem Verhältnis von abgedeckten positiven Beispielen zu allen Instanzen auf die die Regel zutrifft (also dem Konzept der Heuristik *Precision*) und das Verhältnis von positiven Beispielen insgesamt in der Trainingsmenge zu allen vorhandenen Instanzen.

Definition 3.4 (Precision Gain) $g = \frac{p}{p+n} - \frac{P}{P+N}$

Da der hintere Term für alle Regeln konstant ist, entsprechen die Isometriken von *Precision Gain* denen der Heuristik *Precision*. Der Term kommt beim *Klösgen-Maß* hinzu, um eine zu optimistische Abschätzung zu verhindern und um die *a priori*-Verteilung von positiven und negativen Beispielen nicht außer Acht zu lassen. Auf der N-Achse des PN-Raums hat *Precision* den Minimalwert von null, während *Precision Gain* hier den geringsten Wert von $-\frac{P}{P+N}$ hat. Auf der P-Achse des PN-Raums hat *Precision* den Maximalwert von eins, während *Precision Gain* den größten Wert von $1 - \frac{P}{P+N}$ hat. Da die Abdeckung einer Regel von der Gesamtzahl der Beispiele abhängt, muss für einen sinnvollen Vergleich die Anzahl der Instanzen auch bei der Genauigkeitsoptimierung zum Tragen kommen. Das *Klösgen-Maß* verknüpft die beiden Konzepte wie folgt:

$$h_{kloesgen} = c^{\omega} * g = (\frac{p+n}{P+N})^{\omega} * (\frac{p}{p+n} - \frac{P}{P+N}) \tag{3.13}$$

Der Parameter, über den die Präferenz der Heuristik gesteuert werden kann, wird hier mit ω bezeichnet.

Klösgen [17] hat drei verschiedene Kombinationen der beiden Basis-Heuristiken vorgestellt und Wrobel [34] noch eine vierte Variante. Die einzelnen Versionen unterscheiden sich ausschließlich in der Art und Weise wie sie die beiden Basis-Heuristiken kombinieren.

1. $\sqrt{c} * g \rightarrow \omega = 0.5$

Dieses Maß basiert auf der Idee einen statistischen Test über die Verteilung von *Precision Gain* zu machen. Dieser gründet sich auf die Annahme, dass die Genauigkeit der Regel die gleiche ist, wie die Genauigkeit der gesamten Trainingsmenge. In diesem Fall gilt $\frac{p}{p+n} = \frac{P}{P+N}$ und der Wert von *Precision Gain* sollte einer Binomialverteilung rund um null entsprechen.

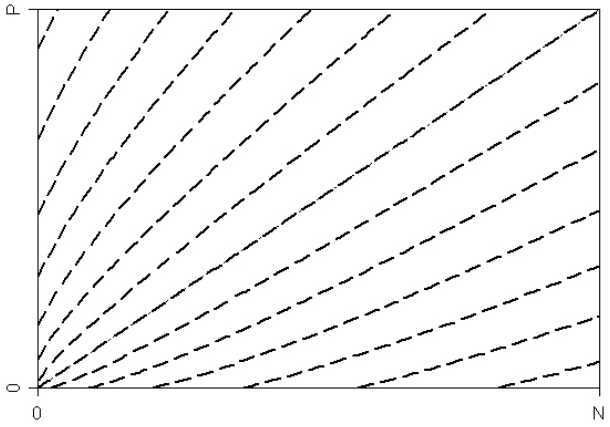

Abbildung 3.13: Isometriken für *Klösgen0.5*

Die resultierende Variante bevorzugt Regeln die nah am Ursprung liegen, was daran zu sehen ist, dass dort die Linien enger beieinander liegen. In dieser Region beginnen sie sich zum Ursprung hin zu biegen. Es ist erkennbar, dass Regeln mit niedriger Abdeckung einen kleineren Abstand von der Diagonale haben als komplexe Regeln, die denselben Evaluationswert zugeordnet bekommen haben und daher von der Heuristik bevorzugt werden.

2. $c * g \rightarrow \omega = 1$

Setzt man den Parameter $\omega = 1$, so erhält man *Weighted Relative Accuracy*. Zum ersten Maß ändert sich nun, dass die Linien parallel sind. Alle Regeln, deren Heuristikwerte denselben Abstand von der Diagonalen haben, werden unabhängig von ihrer Abdeckung gleich evaluiert. Durch diese Erhöhung des Parameters werden nun kleine Regeln nicht mehr bevorzugt und der Einfluss der Abdeckung ist im Vergleich zum vorherigen Maß gesteigert worden.

3. $c^2 * g \rightarrow \omega = 2$

Bei Variante drei wird nun versucht den Einfluss der Abdeckung nochmals zu steigern, indem der Parameter auf zwei erhöht wird.

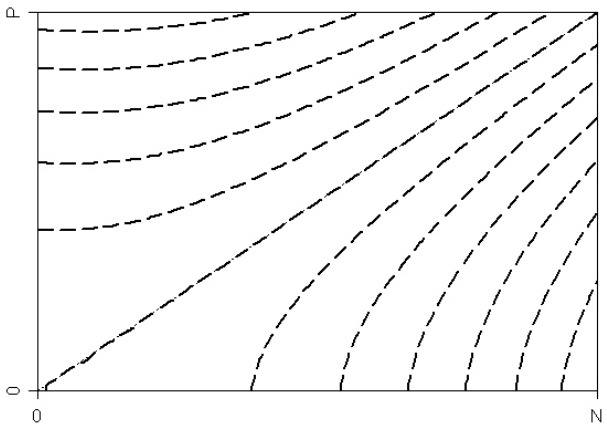

Abbildung 3.14: Isometriken für *Klösgen2.0*

In den Isometriken ist zu erkennen, dass der Bereich nahe dem Ursprung, also Regeln mit niedriger Abdeckung, gemieden wird. Regeln, die mit diesem Maß gefunden werden, tendieren dazu, generell zu sein. Je größer man den Parameter ω wählt, desto stärker macht sich dieser Effekt bemerkbar.

4. $\frac{c}{1-c} * g$

Klösgen hat in [16] gezeigt, dass dieses Maß äquivalent zu einem $\chi^2 - Test$ ist. Da die Heuristik *Correlation* ebenfalls einem $\chi^2 - Test$ entspricht, sind die beiden Isometriken gleich. Wie bei *Correlation* biegen sich die Linien symmetrisch um die Diagonale. Daher werden Regeln mit hoher und niedriger Abdeckung gleich evaluiert, was dieses Maß von den anderen drei Varianten unterscheidet, bei denen entweder Regeln mit niedriger oder mit hoher Abdeckung bevorzugt wurden.

In diesem Kapitel wurden weitere Parametrisierungen des Klösgen-Maßes getestet. Da jede der parametrisierbaren Heuristiken für Werte, die gegen null gehen, zur Basis-Heuristik *Precision* tendiert, ist dieses Verhalten auch beim Klösgen-Maß für $\omega \to 0$ zu erkennen. Wie in Variante eins gezeigt, beginnen sich die Linien bei einer Steigerung des Parameters auf 0.5 in Richtung des Ursprungs zu biegen, während sie in Bereichen mit hoher Abdeckung gegen die Isometrik von *Weighted Relative Accuracy* konvergieren, also parallele Linien ausbilden. Daher bildet das Klösgen-Maß für das Intervall $[0, 1]$ einen Trade-Off zwischen *Precision* und *Weighted Relative Accuracy*.

Die Veränderungen bei kontinuierlicher Erhöhung des Parameters sind in der Abbildung 3.15 zu erkennen. Es fällt ins Auge, dass sich die Linien im Bereich von kleinen Werten von P und N anfangen zu biegen, während sie bei großen Werten eine Tendenz zeigen parallel zu werden. Die Biegung in den Isometriken nimmt mit steigendem Parameter immer weiter ab. Parallel zu dieser Veränderung nähern sich die Steigungen immer weiter an, bis sich die Isometriken von *Weighted Relative Accuracy* ergeben.

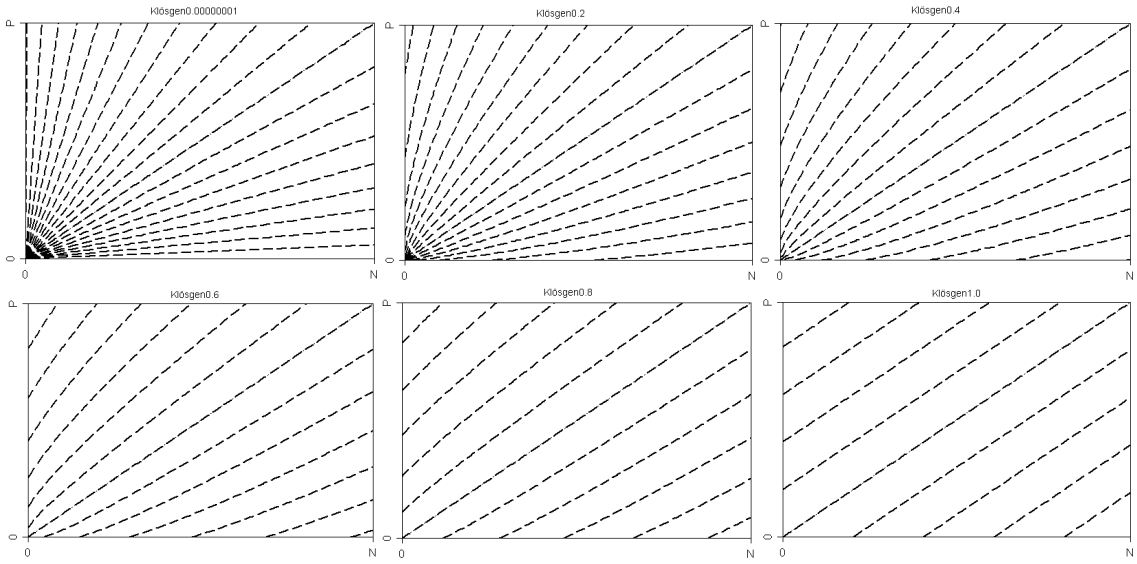

Abbildung 3.15: Isometriken für *Klösgen* zwischen 0 und 1

Im Intervall $[1, \infty)$ wird *Weighted Relative Accuracy* mit *Coverage* verglichen. In Regionen mit niedriger Abdeckung beginnen sich die Linien der Isometrik vom Ursprung weg zu biegen. Dieser Bereich, der eine geringe Abdeckung repräsentiert, wird von der Heuristik gemieden. Betrachtet man die obere linke Ecke der Abbildung 3.16, so ist zu erkennen, dass die Linien sich hier den Isometriken von *Weighted Relative Accuracy* annähern. Die Linien sind dort parallel und befinden sehr nah an der Diagonalen. Die Entwicklung bei weiterer Parametererhöhung ist in den weiteren Abbildungen zu sehen.

Die Linien im Bereich hoher Abdeckung konvergieren in allen Isometriken gegen parallele Linien. Erhöht man den Parameter weiter, so wandern die linken oberen Enden von jeweils zwei dieser parallelen Linien entlang der Diagonale so lange in Richtung des Ursprungs, bis sie sich verbinden, wie in den beiden Abbildungen für die Erhöhung des Parameters von 30 auf 500 zu sehen ist. Die resultierende Linie befindet sich dann rechtwinklig zur 45°-Steigung. Daraus entstehen die Isometriken von *Coverage* für $\omega \to \infty$. Dass auf der Diagonalen ebenfalls verschiedene Isometriken zu sehen sind, kann hier ignoriert werden[4]. Interessant ist die Beobachtung, dass sich die Evaluationswerte stark verändern. Geht man von der Linie aus, die sich im linken Teil des Dreiecks, welches von der Diagonalen gebildet wird, und am weitesten unten befindet (diese Linie entspricht der niedrigsten, positiven Evaluation, da alle Linien rechts von der Diagonalen negative Werte haben), so verändert sich der Wert drastisch. Bei einem Parameter von zwei entspricht diese Linie einem Evaluationswert von 0.02. Bereits bei einem Parameter von sieben ist der Wert $1 * E^{-6}$. Er fällt weiter auf $1 * E^{-20}$ bei einer Parametrisierung von 30. Setzt man den Parameter schließlich auf 500, so entspricht die Linie einem Evaluationswert von $1 * E^{-320}$.

[4]Es handelt sich hierbei um von den sichtbaren Linien disjunkte Evaluationswerte, die der Übersicht halber nicht mit gezeichnet wurden.

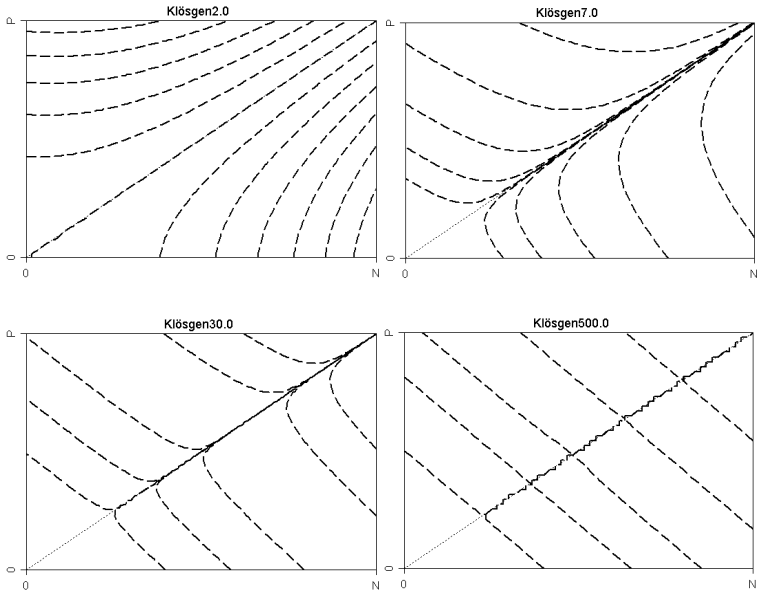

Abbildung 3.16: Isometriken für *Klösgen* zwischen 1 und ∞

3.4.2 Das m-Estimate

Bei der Heuristik *m-Estimate* geht man wie bei *Laplace* davon aus, dass jede Regel eine gewisse Anzahl an Beispielen *a priori* abdeckt. Es ist äquivalent zur Heuristik *Precision*, nur dass man nicht bei null anfängt die positiven und negativen Beispiele zu zählen.

$$h_{m\_Estimate} = \frac{p+m*\frac{P}{P+N}}{p+n+m} \tag{3.14}$$

Geht man von einer Gleichverteilung der positiven und negativen Beispielen aus und setzt den Parameter $m = 2$, so erhält man die Heuristik *Laplace*, den Standard-Fall des *m-Estimates*.

$$h_{lap} = h_{m-estimate}(2.0) = \frac{p+2.0*(\frac{P}{P+N})}{p+n+2.0} = \frac{p+2.0*\frac{1}{2}}{p+n+2.0} = \frac{p+1}{p+n+2} \tag{3.15}$$

Dies entspricht der gängigsten Parametrisierung dieser Heuristik. Beim *m-Estimate* hingegen nimmt man eine *a priori*-Verteilung von $\frac{P}{P+N}$ der Beispiele in der Trainingsmenge an und kann mit dem Parameter m angeben, wie viele Beispiele jede Regel im Vorhinein abdecken soll. Im PN-Raum resultiert dies in einer Verschiebung des Ursprungs zum Punkt $(-n_m, -p_m)$, wobei für *Laplace* $n_m = p_m = 1$ und für das *m-Estimate* $n_m = m - p_m$, mit $p_m = m * \frac{P}{P+N}$ gilt. Die Diagonale beginnt bei diesem Punkt, schneidet dann den Ursprung und endet im Punkt (N, P). Da man bei *Laplace* von einer Gleichverteilung der Beispiele ausgeht, sind hier die Isometriken symmetrisch zur 45°-Steigung, während sie sich beim *m-Estimate* symmetrisch zur Diagonalen befinden.

Beim *m-Estimate* wird ein Trade-Off zwischen *Precision* und *Weighted Relative Accuracy* gebildet. Da *Weighted Relative Accuracy* den Lernalgorithmus zum Punkt $(0, P)$ lenkt,

wird beim *m-Estimate* für große Parameterwerte nicht so sehr auf eine reine Abdeckung abgezielt. Das ist auch der größte Unterschied zu den anderen beiden parametrisierbaren Heuristiken. Wählt man den Parameter $m = 0$, so erhält man Precision. Gilt hingegen $m \to \infty$, so konvergieren die Isometriken gegen parallele Linien mit der Steigung

$$\frac{1-c}{c}, \tag{3.16}$$

wie in [10] gezeigt wurde. Da die Linien bei der Heuristik *Weighted Relative Accuracy* parallel zur Diagonalen des PN-Raumes sind, ist ihre Steigung $\frac{P}{N}$, wie aus Tabelle 3.1 folgt. Wie in Abschnitt 3.3.3 aufgezeigt worden ist, wird die Konstante der Kostenfunktion wie folgt gesetzt, wenn man die Isometriken der Heuristik erhalten möchte: $c = \frac{N}{P+N}$. Setzt man dies in die Steigung für $m \to \infty$ aus (3.15) ein, so folgt $\frac{1-c}{c} = \frac{1-\frac{N}{P+N}}{\frac{N}{P+N}} = \frac{\frac{P}{P+N}}{\frac{N}{P+N}} = \frac{P}{N}$. Die Isometriken für den Parameter $m \to \infty$ des *m-Estimates* konvergieren also gegen die von *Weighted Relative Accuracy*. In der Grafik 3.17 aus [10] kann man den Effekt der Verschiebung des Ursprungs erkennen.

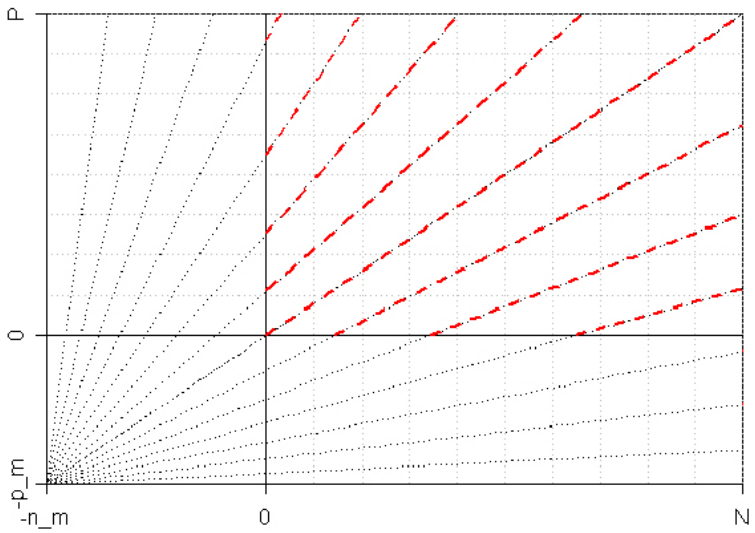

Abbildung 3.17: allgemeines Schema des *m-Estimates*

Verschiebt man nun den Punkt $(-n_m, -p_m)$ immer weiter vom Ursprung weg, so werden die Linien der Isometrik mehr und mehr parallel, bis sie gegen *Weighted Relative Accuracy* konvergieren. Man kann in Abbildung 3.18 erkennen, dass die Veränderungen in den Isometriken von *Precision* und *Weighted Relative Accuracy* sehr gering sind. Aus diesem Grund sollte man davon ausgehen, dass sich beim *m-Estimate* die Parameterwerte mit einer hohen Genauigkeit über ein größeres Intervall erstrecken, als beim *Klösgen-Maß* oder beim *F-Measure*.

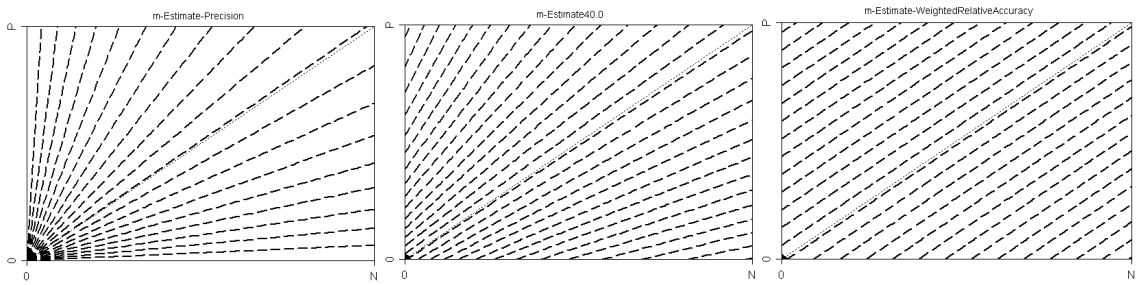

Abbildung 3.18: Isometriken für das *m-Estimate* zwischen 0 und ∞

Des weiteren ist *Weighted Relative Accuracy* eine Heuristik mit besserer durchschnittlicher Genauigkeit als *Precision*, weshalb für große Parameterwerte genauere Ergebnisse als für kleine erzielt werden, was ebenfalls nur beim *m-Estimate* der Fall war. Auf die Probleme, die sich bei dieser Heuristik aus dem Trade-Off mit der untypischen Basis-Heuristiken ergeben, wird in Abschnitt 5.1.2 genauer eingegangen.

3.4.3 Das F-Measure

Die Heuristik *F-Measure* [25] kommt aus dem *Information Retrieval* und bildet einen Trade-Off zwischen *Precision* und *Recall*.

$$h_{f\_Measure} = \frac{(\beta^2+1)*Precision*Recall}{\beta^2*Precision+Recall} \tag{3.17}$$

Im *Information Retrieval* beschreibt der *Recall* den Anteil von erhaltenen relevanten Dokumenten von allen Dokumenten und die *Precision* den Anteil relevanter Dokumente von allen erhaltenen Dokumenten. Üblicherweise kann man den *Recall* steigern, indem man die *Precision* senkt und umgekehrt. Es sind verschiedene Parametersetzungen des *F-Measure* gängig:

- $\beta = 0$ *(Precision)*
- $\beta = 0.5$ *(Precision* ist wichtiger als *Recall)*
- $\beta = 1$ *(Precision* und *Recall* werden gleich gewichtet)
- $\beta = 2$ *(Recall* ist wichtiger als *Precision)*
- $\beta \to \infty$ *(Recall)*

Für den Fall $\beta = 1$ ist das *F-Measure* außerdem ein harmonisches Mittel[5].

Definition 3.5 (Harmonisches Mittel) $\frac{1}{H} = \frac{1}{n} * \sum_{i=1}^{n} \frac{1}{x_i}$ mit

[5]Fährt man beispielsweise 100 km mit 50 km/h und 100 km mit 100 km/h, so legt man 200 km in drei Stunden zurück und die Durchschnittsgeschwindigkeit beträgt 66.67 km/h, eben genau das harmonische Mittel von 50 und 100. Dieses bezieht sich auf die benötigte Zeit. Das mit der benötigten Zeit gewichtete arithmetische Mittel der Teilgeschwindigkeiten wäre äquivalent.

- n ist in diesem Beispiel die Anzahl der Heuristiken und
- x_i ist die *i-te* Heuristik

Wendet man diese Formel auf *Precision* und *Recall* an, so erhält man

$$H = \frac{1}{\frac{1}{2}*(\frac{1}{Recall}+\frac{1}{Precision})} = \frac{2}{\frac{Recall}{Recall*Precision}+\frac{Precision}{Recall*Precision}} = \frac{2*Recall*Precision}{Recall+Precision}. \tag{3.18}$$

Diese Parametrisierung entspricht zusätzlich dem Standard-Fall des *F-Measure*s.

Beim Regel-Lernen sind *Precision* und *Recall* wie in den zugehörigen Abschnitten 3.3.1 und 3.3.2 definiert. In den Isometriken ist ein ähnlicher Effekt wie beim *m-Estimate* zu erkennen, nur dass sich der Ursprung nicht nach unten links weg bewegt, sondern auf der negativen N-Achse des PN-Raums. Je weiter sich der Punkt $(-g, 0)$ vom Ursprung weg bewegt, desto mehr werden die Linien der Isometrik parallel, bis sie dann für $\beta \to \infty$ gegen *Recall* konvergieren. In der Abbildung 3.19 aus [11] ist dieser Effekt erkennbar.

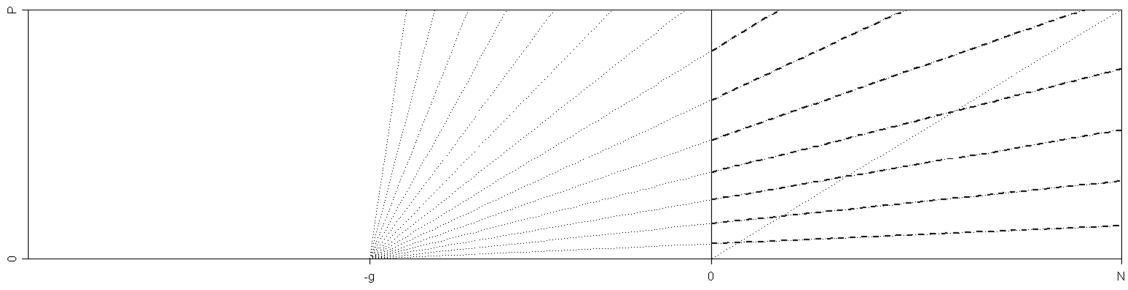

Abbildung 3.19: allgemeines Schema des *F-Measures*

In der Abbildung 3.20 sind die Isometriken für $\beta \to 0$, $\beta = 0.5$, $\beta = 2$ und $\beta \to \infty$ aufgetragen. Der Effekt der Verschiebung des Ursprungs äußert sich darin, dass in den oberen rechten Isometriken die Linien unterhalb der unteren Linie nahezu unverändert bleiben. Im linken oberen Ausschnitt der Grafik ist zu erkennen, dass die Steigung der Linien geringer ist, als für $\beta \to 0$. Betrachtet man sich die beiden unteren Isometriken, so wird dieser Effekt noch verstärkt. Die Steigung der Linien im unteren Bereich nimmt langsam immer weiter ab, während die der Linien im oberen linken Ausschnitt sehr viel schneller geringer wird. Erhöht man den Parameter weiter, so wird die Steigung aller Linien immer geringer. Unterhalb der Abtrennungslinie ist die Steigung bereits bei *Precision* gering, weshalb hier die Verringerung nicht so deutlich zu sehen ist, wie im oberen Ausschnitt. Lässt man den Parameter gegen ∞ laufen, so ergeben sich die Isometriken der Basis-Heuristik *Recall*, wie im unteren, rechten Teil der Grafik zu sehen ist.

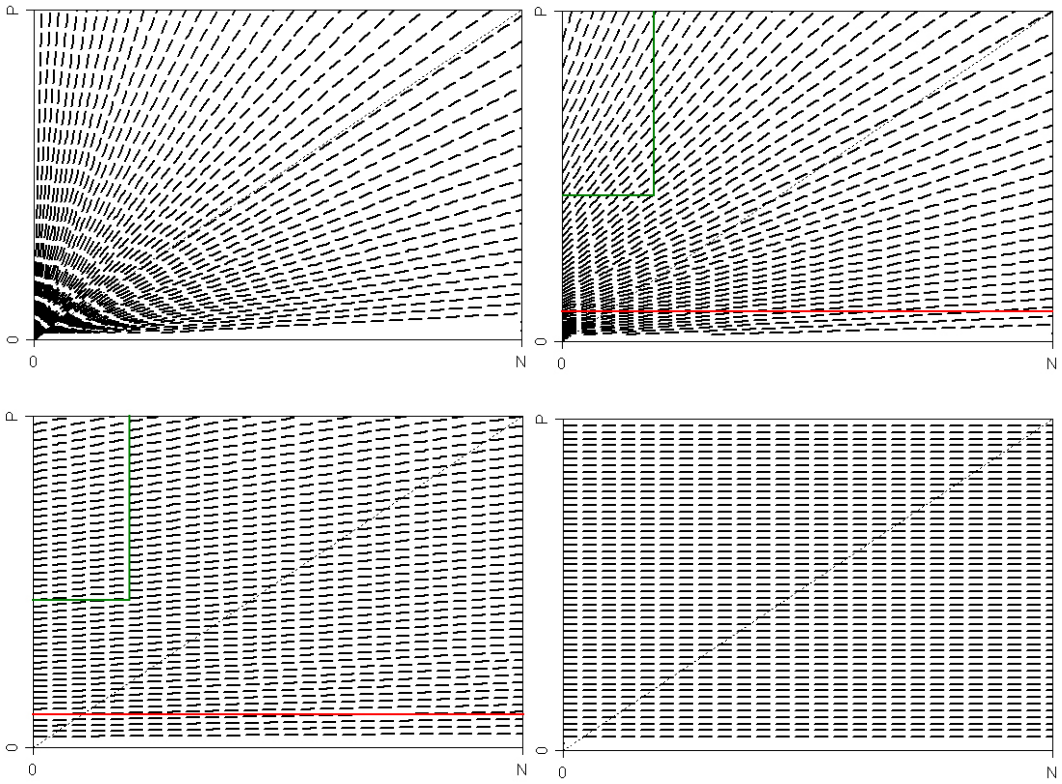

Abbildung 3.20: Isometriken für das *F-Measure* zwischen 0 und ∞

Kapitel 4

Experimentelles Szenario

In diesem Kapitel wird beschrieben, welche Eckdaten für die verschiedenen Experimente gegolten haben. Begonnen wird mit einer Beschreibung der Datenmengen. Es folgt eine detaillierte Zusammenfassung der Methode Cross-Validation und der verwendete Regel-Lerner wird näher erläutert. Außerdem werden die verschiedenen Methoden zur Evaluation erklärt und deren Vor- und Nachteile aufgezeigt.

4.1 Beschreibung der UCI-Datenmengen

Für die Suche nach den optimalen Parametern wurden 27 Datenmengen aus dem UCI-Repository [20] verwendet. Das UCI-Repository ist eine große Menge von Daten, die beim maschinellen Lernen häufig zum Einsatz kommen. Alle Datenmengen lagen im .arff-Format vor, da der Regel-Lerner nur mit diesem Format umgehen konnte. Die Tabelle 4.1 gibt eine kurze Beschreibung der wichtigsten Eigenschaften der einzelnen Datenmengen.

Zum Testen der Parameter wurden weitere 30 Datenmengen verwendet. Diese stammen ebenfalls aus dem UCI-Repository. Auch ihre Eigenschaften sind in Tabelle 4.2 kurz zusammengefasst.

4.2 Der verwendete Regel-Lerner

Es ist der Regel-Lerner aus dem SeCo-Framework [28] verwendet worden. Der Vorteil dieses Frameworks ist sein modularer Aufbau. So ist es zum Beispiel möglich, das Suchverfahren, die Suchheuristik oder auch die Methodik zur Vermeidung von *Overfitting* auszutauschen. Für diese Arbeit war es vor allem wichtig, dass die Suchheuristiken frei wählbar waren. Als Suchverfahren wurde *Top-Down Hill-Climbing* verwendet. Da der Lernalgorithmus für alle

Name	# Beispiele	# Attribute	# Klassen
anneal	798	38	6
audiology	226	69	24
breast-cancer	286	9	2
cleveland-heart-disease	303	13	5
contact-lenses	24	4	3
credit	490	15	2
glass2	163	9	2
glass	214	9	7
hepatitis	155	19	2
horse-colic	368	22	2
hypothyroid	3163	25	2
iris	150	4	3
krkp	3196	36	2
labor	57	16	2
lymphography	148	18	4
monk1	124	6	2
monk2	169	6	2
monk3	122	6	2
mushroom	8124	22	2
sick-euthyroid	3163	25	2
soybean	683	35	19
tic-tac-toe	958	9	2
titanic	2201	3	2
vote-1	435	15	2
vote	435	16	2
vowel	990	9	11
wine	178	13	3

Tabelle 4.1: Die 27 UCI-Datenmengen

Heuristiken gleich instantiiert wurde, war es unerheblich, welche Suchstrategie eingesetzt worden ist. Es wurde die einfachste Abbruchbedingung benutzt. Daher werden alle möglichen Regeln erstellt und die beste ausgewählt. Um die verschiedenen Heuristiken zu testen, wurde eine Methode *TestHeuristics* implementiert, die die Datenmengen und eine Liste der zu verwendenden Heuristiken mit ihren Parametern als Eingabe erhält. Ausgegeben wird die Genauigkeit der einzelnen Heuristiken pro Datenmenge. Diese Daten werden mit den Evaluationsmechanismen von *weka* [33] mit einer stratifizierten 1x10 Cross-Validation (Standard-Parameter) erstellt. Mit verschiedenen Skripten wird dann die durchschnittliche Genauigkeit mit einer der folgenden Evaluationsmethoden berechnet.

Name	# Beispiele	# Attribute	# Klassen
auto-mpg	398	7	4
autos	205	25	7
balance-scale	625	4	3
balloons	76	4	2
breast-w	699	9	2
breast-w-d	699	9	2
bridges2	105	11	6
colic	368	22	2
colic.ORIG	368	27	2
credit-a	690	15	2
credit-g	1000	20	2
diabetes	768	8	2
echocardiogram	74	8	2
flag	194	27	6
hayes-roth	132	4	3
heart-c	303	13	5
heart-h	294	13	5
heart-statlog	270	13	2
house-votes-84	435	16	2
ionosphere	351	34	2
labor-d	57	16	2
lymph	148	19	4
machine	209	10	8
primary-tumor	339	17	22
promoters	106	57	2
segment	2310	19	7
solar-flare	333	10	8
sonar	208	60	2
vehicle	846	18	4
zoo	101	17	7

Tabelle 4.2: Die 30 UCI-Datenmengen

4.3 Evaluation

Um die erzielten Ergebnisse zu bewerten, wurden verschiedene Möglichkeiten zur Evaluation verwendet. Ein generelles Problem beim Bilden des Durchschnitts mit Macro-Average-Accuracy liegt in Ausreißern, bei denen die Heuristik auf einer Datenmenge sehr schlecht abschneidet. Erreicht sie sonst ausschließlich gute Werte, verschlechtert sich die durchschnittliche Genauigkeit durch diese wenigen Datenmengen rapide. Dieses Problem tritt bei der Bewertung von Ergebnissen im maschinellen Lernen öfters auf. Häufig ist man versucht die betreffenden Datenmengen aus den Resultaten zu entfernen, um höhere Genauigkeiten zu erzielen. Solange aber alle Heuristiken schlechte Ergebnisse auf diesen Datenmengen erreichen, gleicht sich der Effekt wieder aus. Trotzdem ist es sinnvoll, unterschiedliche Eva-

luationsmethoden zur Bewertung der Resultate zu verwenden. Eine Möglichkeit besteht darin, die Datenmengen unterschiedlich zu gewichten, also beispielsweise kleinen Mengen ein niedrigeres Gewicht zuzuordnen als großen. Bei der Evaluation ist aufgefallen, dass sich die Genauigkeiten bei großen Datenmengen nicht so stark unterscheiden wie bei kleinen. Da alle Heuristiken auf diesen Mengen sehr gute Ergebnisse erzielen, ist es sinnvoll, eine Methode einzuführen, die die Unterschiede deutlicher aufzeigt. Dieser Ansatz, der die Abweichungen in der Genauigkeit der einzelnen Heuristiken objektiver bewertet, besteht in Rankingverfahren. Mit einem Ranking wird versucht, den Heuristiken bestimmte Ränge zu erteilen, die die Güte der verschiedenen Methoden in einer anderen Form als der durchschnittlichen Genauigkeit widerspiegeln. Erreicht beispielsweise eine Heuristik im Durchschnitt auf allen Datenmengen 90% Genauigkeit und neun andere Maße bekommen 89.9%, so ist die Bewertung mit Macro-Average-Accuracy nahezu gleich (siehe Tabelle 4.4). Mit einem guten Rankingverfahren ist es hingegen möglich die Unterschiede in der Genauigkeit der verschiedenen Heuristiken besser aufzuzeigen. Die Rankings sind außerdem wichtig um einen Korrelationskoeffizienten berechnen zu können, der angibt wie sehr sich zwei verschiedene Rankings unterscheiden.

4.3.1 Cross-Validation

Ein Problem bei der Bewertung der Güte eines Klassifikators tritt dann auf, wenn ausschließlich Trainingsbeispiele und keine expliziten Testbeispiele vorhanden sind. Eine Möglichkeit trotzdem die Genauigkeit des Klassifikators auf ungesehenen Daten einzuschätzen, besteht in Kreuzvalidierungsverfahren (Cross-Validation).

k-fold Cross-Validation Bei diesem Verfahren wird vor dem Lernprozess die Trainingsmenge T, die aus n Instanzen besteht, in k zufällige, ungefähr gleich große Teile aufgespalten, wobei $k<n$ gilt. Die einzelnen Teile werden als *folds* bezeichnet. Nun wird der Klassifikator k-mal trainiert. Dabei wird jeweils die k-te Teilmenge als Testmenge und die verbleibenden *k-1* Teilmengen als Trainingsmenge verwendet. Es wird für jeden Durchlauf die Genauigkeit des Klassifikators auf der jeweiligen Testmenge berechnet. Die Gesamtgenauigkeit ergibt sich aus dem Durchschnitt der einzelnen Genauigkeiten. In verschiedenen Tests [18] hat sich ein optimales k von 10 ergeben. Das Problem dieser Methode liegt darin, dass die durchschnittliche Genauigkeit von der zufälligen Aufteilung der Daten abhängig ist. Eine Lösung dieses Sachverhalts wäre beispielsweise, die Cross-Validation mehrfach mit unterschiedlichen Aufteilungen der Daten laufen zu lassen. Man spricht in diesem Fall von einer *a x b-Cross-Validation*, wobei a angibt, wie oft unterschiedliche Aufteilungen gewählt wurden und b notiert, in wie viele Teile die Daten aufgeteilt wurden. Der Standard wäre in dieser Notation eine 1x10-Cross-Validation.

stratified Cross-Validation Diese Methode funktioniert analog zur *k-fold Cross Validation,* nur dass darauf geachtet wird, in den einzelnen Teilen (folds) die Klassenverteilung der Ausgangsdatenmenge zu erhalten.

leave-one-out Cross-Validation Hier wird k=n gesetzt. Der Nachteil dieser Methode ist, dass eine Aufteilung, die die Klassenverteilung aufrecht erhält, nun unmöglich ist. Das kann zu führen, dass falsche Genauigkeiten ermittelt werden, wenn die Beispiele der Datenmenge ungünstig gewählt wurden. Sind beispielsweise nur zwei Klassen vorhanden, die gleichmäßig verteilt sind und vollkommen zufällig in der Datenmenge vorkommen, so würde die Genauigkeit mit dieser Methode immer 0 % betragen. Das liegt daran, dass zum Trainieren immer eine Aufteilung verwendet wird, in der die Klasse, die vorhergesagt werden soll, am seltensten ist. Daher wird ein Klassifikator, der auf einer Mehrheitswahl basiert, die andere Klasse vorhersagen und immer fehlerhaft klassifizieren. Ein weiterer Nachteil ist die hohe Rechenzeit, die bei dieser Methode benötigt wird.

4.3.2 Macro Average Accuracy

Es gibt verschiedene Möglichkeiten die Güte einer Heuristik zu bestimmen. Hat man die Genauigkeit pro Datenmenge und möchte die durchschnittliche Genauigkeit auf allen Mengen berechnen, so bietet sich die Evaluationsmethode Macro-Average-Accuracy an. Hierbei wird das arithmetische Mittel gebildet:

Definition 4.1 (Macro Average Accuracy)

$$Average\ Accuracy_{macro} = \frac{\sum_{i=1}^{Gesamtanzahl\ Datenmengen} \frac{korrekt\ klassifizierte\ Beispiele_i}{Gesamtanzahl\ Beispiele_i}}{Gesamtanzahl\ Datenmengen}$$

Bei dieser Methode werden alle Datenmengen gleich gewichtet. Daher spielen hier Ausreißer eine große Rolle. Erreicht eine Heuristik z.B. auf neun Datenmengen jeweils 98% Genauigkeit und nur auf einer einzigen 50%, so bekommt sie mit Macro-Average-Accuracy eine durchschnittliche Genauigkeit von 93.2%. Um dieser Situation entgegenzuwirken könnte man die einzelnen Datenmengen unterschiedlich gewichten. Dieser Ansatz wird bei Micro-Average-Accuracy versucht.

4.3.3 Micro Average Accuracy

Bei Micro-Average-Accuracy werden erst alle korrekt klassifizierten Beispiele aufsummiert und dann durch die Gesamtanzahl der Beispiele geteilt. Daraus resultiert der Effekt, dass große Datenmengen höher gewichtet werden als kleine. Je kleiner die Anzahl der Beispiele ist, desto weniger fällt diese Datenmenge ins Gewicht.

Zeile	Datenmenge	# Beispiele	# korrekt klassifizierte Beispiele	Macro-Accuracy	Micro-Accuracy
1	A	550	501	91.09	
2	B	270	235	87.04	
3	C	320	311	97.19	
4	D	51	49	96.08	
5	E	110	99	90.00	
6	F	690	669	96.96	
7	G	150	135	90.00	
8		2141	1999	92.62	93.37
9	H	15	8	55.33	
10		2156	2007	87.71	93.09
11	I	500	267	53.4	
12		2641	2266	87.72	85.8
13	J	1000	551	55.1	
14		3141	2550	87.94	81.18

Tabelle 4.3: Unterschiede zwischen Macro- und Micro-Average-Accuracy

Definition 4.2 (Micro Average Accuracy)

$$Average\ Accuracy_{micro} = \frac{\sum_{i=1}^{Gesamtzahl\ Datenmengen} korrekt\ klassifizierte\ Beispiele_i}{\sum_{i=1}^{Gesamtzahl\ Datenmengen} Gesamtanzahl\ Beispiele_i}$$

Die Tabelle 4.3 gibt ein Beispiel, um die Unterschiede zwischen Macro- und Micro-Average-Accuracy aufzuzeigen. Beide Genauigkeitsbewertungen sind jeweils in Prozent angegeben.

Die Zeilen eins bis sieben repräsentieren die Ausgangs-Datenmenge. In der achten Zeile stehen die Gesamtanzahlen der Beispiele und die der korrekt klassifizierten Beispiele. Außerdem ist die Macro- und die Micro-Average-Accuracy aufgetragen. Fügt man nun die Datenmenge H hinzu (die neunten Zeile der Tabelle), die über 15 Beispiele verfügt, von denen acht korrekt klassifiziert wurden, erkennt man, dass die Macro-Average-Accuracy auf 87.71% abfällt, während sich die Micro-Average-Accuracy nur leicht verschlechtert. Entfernt man die Datenmenge H wieder und fügt die Menge I hinzu, die über deutlich mehr Beispiele verfügt, fällt die Micro-Average-Accuracy knapp unter den Wert der Macro-Average-Accuracy. Fügt man den sieben Ausgangdatenmengen ausschließlich die Menge J hinzu, die am meisten Beispiele enthält, wird dieser Effekt nochmals verstärkt. Man sieht, dass die Micro-Average-Accuracy auf schlecht klassifizierte große Datenmenge sehr sensibel reagiert, sie also höher gewichtet. Die Macro-Average-Accuracy hat durch das Hinzufügen der Datenmengen mit geringer Genauigkeit nahezu denselben durchschnittlichen Wert.

4.3.4 Ranking

Eine andere Möglichkeit zur Bewertung der Güte einer Heuristik ist eine Aufsummierung aller Gewinne auf den einzelnen Datenmengen. Dieses Verfahren wird ebenfalls häufig im

theoretischer Rang	Heuristik	Genauigkeit in %	Ranking
1	Heur1	95	$\frac{1+2+3}{3} = 2$
2	Heur2	95	2
3	Heur3	95	2
4	Heur4	92	4
5	Heur5	88	$\frac{5+6}{2} = 5.5$
6	Heur6	88	5.5

Heuristik	Genauigkeit in %	Ranking
Heur1	90	1
Heur2	89.9	6
Heur3	89.9	6
Heur4	89.9	6
Heur5	89.9	6
Heur6	89.9	6
Heur7	89.9	6
Heur8	89.9	6
Heur9	89.9	6
Heur10	89.9	6

Tabelle 4.4: Beispiel zum Ranking und Vergleich mit Genauigkeitsabschätzungen

Maschinellen Lernen verwendet. Sobald eine Heuristik die höchste Genauigkeit auf einer bestimmten Datenmenge erreicht hat, ist sie auf dieser Menge der Gewinner. Ordnet man allen anderen Heuristik ebenfalls eine Position innerhalb der Menge relativ zu allen Heuristiken zu, so erhält man eine Liste mit Positionen. Diese Rangliste gibt pro Datenmenge an, wie gut eine bestimmte Heuristik im direkten Vergleich mit allen anderen ist. Der Vorteil liegt darin, dass man nicht mehr an eine gewisse Prozentzahl gebunden ist. Liegen die Genauigkeiten der Heuristiken nah beieinander oder sind gleich, ist der Vergleich mit Macro-Average-Accuracy schwierig, da die Unterschiede so gering sind. Für diesen Fall bietet sich ein Ranking an. In der Rangliste hat jede Heuristik eine bestimmte Position, die unabhängig von den Differenzen der Genauigkeiten ist. So erhält beispielsweise eine Heuristik die Position drei und eine andere den vierten Platz, obwohl ihre Genauigkeit sich nur um 0.00001 % unterscheidet. Die Berechnung der Position in der Rangliste ist in Tabelle 4.4 skizziert. Es gibt verschiedene Möglichkeiten eine Rangliste zu erstellen. Bei sportlichem Wettkampf würde es nach der Situation in der linken Tabelle drei erste Plätze, einen zweiten Platz und zwei dritte Plätze geben. Eine anderes Verfahren würde zum Beispiel ebenfalls allen drei besten Heuristiken den ersten Platz zuordnen, der nächst schlechteren allerdings den theoretischen Rang erteilen. Daher hätte in diesem Fall *Heur4* nicht mehr den zweiten, sondern den vierten Platz.

Um einen Korrelationswert zweier Ranglisten zu berechnen, wird häufig die Spearman Rank Correlation benutzt. Diese Korrelationsberechnung fordert ein spezielles statistisches Rankingverfahren, was wie folgt definiert ist.

Definition 4.3 (Rang) $Rang = \frac{\sum_{theoretischer\ Rang}}{Anzahl\ Heuristiken\ mit\ gleicher\ Genauigkeit}$

Der Vorteil dieses Rankingmechanismus liegt in der objektiveren Einschätzung einer Heuristik. Angenommen man hat die Situation aus Tabelle 4.4 (rechts). Die Genauigkeitsabschätzung wurde entweder mit Micro- oder mit Macro-Average-Accuracy erstellt. Der tatsächliche, eklatante Unterschied zwischen der ersten und allen anderen Heuristiken kommt

allerdings bei der Genauigkeitsabschätzung nicht zum Vorschein. Er fällt erst dann auf, wenn man den Rankingmechanismus verwendet.

4.3.4.1 Spearman Rank Correlation

Der Spearman Rank Correlation Koeffizient gibt an, wie stark zwei Variablen, die mindestens auf einer Ordinalskala gemessen wurden, korrelieren. Eine Ordinalskala hat als ergänzende Eigenschaften zu einer Nominalskala, auf der die Relationen $=$ und $\neq$ definiert sind, noch die beiden Komparatoren $<$ und $>$. Außerdem gilt zusätzlich die Transitivität. Es handelt sich um ein parameterfreies Maß, das seinen Ursprung in der Statistik hat. Unter dem Begriff parameterfrei, der synonym mit nicht-parametrisch verwendet wird, versteht man ein Modell, in welchem die Struktur nicht *a priori* festgelegt ist, sondern durch die Beschaffenheit der Daten erst entsteht. Der Spearman Rank Correlation Koeffizient eignet sich auch für Variablen deren Beziehung nichtlinear sind und ist robust gegenüber Ausreißern. Das Ergebnis bewegt sich zwischen -1 und 1, wobei der erste Wert eine perfekte negative Korrelation repräsentiert und der zweite eine perfekte positive. Ist das Ergebnis null, so liegt keinerlei Korrelation vor.

Definition 4.4 (Spearman Rank Correlation) $\rho = 1 - \frac{6 * \sum D^2}{N * (N^2 - 1)}$ mit

- D entspricht der Differenz zwischen den Rängen der beiden Variablen und
- N der Anzahl der Wertpaare.

Da Verfahren, die die Korrelation nicht anhand von Rängen ermitteln, häufig gewisse Ungenauigkeiten aufweisen, hat es sich angeboten, die Spearman Rank Correlation zu verwenden. Nötig ist sie in Abschnitt 5.4, um die Güte der gefundenen Parameter für neue Datenmengen abzuschätzen.

Kapitel 5

Resultate

In diesem Kapitel wird die experimentelle Bestimmung von optimalen Parametern für die drei parametrisierbaren Heuristiken beschrieben. Mit optimal ist gemeint, dass die Macro-Average-Accuracy auf allen Datenmengen mit den gefundenen Werten am höchsten ist. Im vorherigen Kapitel ist deutlich geworden, dass eine Optimierung anhand von anderen Evaluationsmethoden unterschiedliche beste Parameter geliefert hätte. Zur Bestimmung der Parameter wurde eine Art manuelles Data-Mining gemacht, indem „per Hand“ verschiedene Parametrisierungen gewählt und getestet wurden.

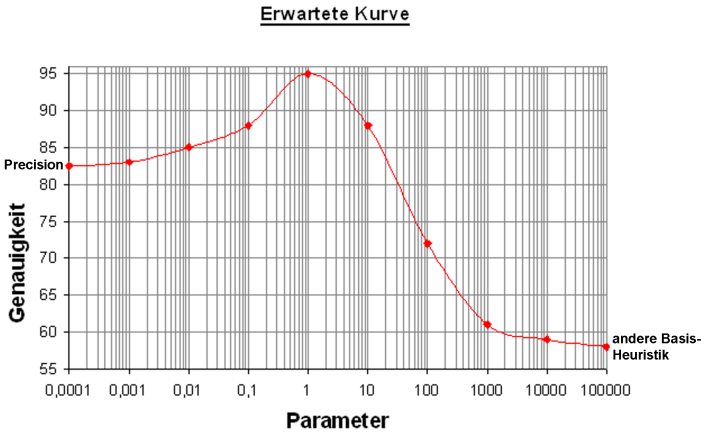

Abbildung 5.1: Erwartete Kurve

Man sollte davon ausgehen, dass sich für die Parametrisierungen eine Kurve ergibt, die bei einem geringen Parameterwert gegen die Genauigkeit von *Precision* konvergiert und mit Erhöhung des Parameterwertes immer genauer wird. Dieser Prozess wird solange voranschreiten, bis ein Optimum gefunden ist. Bei weiterer Erhöhung des Parameters müsste die Genauigkeit immer weiter abnehmen, bis sie für große Werte schließlich gegen die Basis-Heuristik konvergiert, die die Abdeckung repräsentiert. Wenn der Parameter der Heuristiken null ist, konvergieren alle gegen *Precision*. In der Grafik ist zu erkennen, dass die Genauigkeit für große Parameter geringer ist, als für kleine. Das ist dann der Fall, wenn die Prozentzahl der richtig klassifizierten Beispiele von *Precision* größer ist, als die der anderen Basis-Heuristik. Beim *m-Estimate*, welches für große Parameter gegen *Weighted Relative*

Accuracy konvergiert, tritt genau der umgekehrte Fall ein, da hier die Basis-Heuristik für große Werte eine höhere Genauigkeit als *Precision* erzielt.

5.1 Die Bestimmung der optimalen Parameter

Es ist am Anfang unklar gewesen, wie die einzelnen Heuristiken sich verhalten. Deshalb sind in einer ersten Testserie sehr allgemeine Werte benutzt worden, um herauszufinden, welche Parametrisierungen überhaupt interessant sind. Diese Serie hat aus diesem Grund aus einem sehr kleinen Parameter nahe bei null, gefolgt von einem Bereich mit konstanter Schrittweite von eins und einem sehr großen Wert bestanden. Der Durchlauf diente einem groben Überblick. Zu erwarten ist, dass sich der kleine Parameter der Genauigkeit von *Precision* und der große Wert in etwa der der anderen Basis-Heuristik annähert. Außerdem sollte sich im Intervall mit konstanter Schrittweite ein bester Parameter ergeben. Da sowohl beim *F-Measure* als auch beim *Klösgen-Maß* die Basis-Heuristiken, die zur Messung der Abdeckung verwendet werden, eine geringe Genauigkeit besitzen, ist es nahe liegend, dass hier größere Parametrisierungen bereits sehr schlechte Ergebnisse erzielen. Bei beiden Heuristiken wird zum Beispiel mit einem Wert von 10 die Abdeckung bereits sehr viel stärker gewichtet als die Genauigkeit. Daher ist als Intervall mit konstanter Schrittweite der Bereich zwischen 1 und 10 verwendet worden. Zu erwarten ist, dass sich das *m-Estimate* anders verhält, da hier die Genauigkeit der Abdeckungsheuristik deutlich größer ist. Im ersten Testlauf hat sich dann für die Parameter folgende Tabelle ergeben.

Parameter	Genauigkeit in %		
	F-Measure	Klösgen-Maß	m-Estimate
0.000001	82.4956	82.4956	82.4956
1	79.2174	82.7405	82.9757
2	75.1614	68.3501	83.4355
3	73.2423	61.6183	83.7440
4	70.6572	59.1542	84.1491
5	69.6803	58.3782	83.9729
6	69.0166	57.8327	84.6252
7	67.9591	57.4094	85.0888
8	67.5604	57.2956	85.2046
9	67.4433	57.2956	85.2003
10	67.3479	57.2531	85.1746
500	67.2064	55.8597	83.9561

Tabelle 5.1: Erster Testlauf

Das *Klösgen-Maß* hat seine höchste Genauigkeit beim Parameter eins und das *F-Measure* bei 0.000001. Man kann davon ausgehen, dass beide Heuristiken mit anderen Parametrisierungen höhere Genauigkeiten erreichen können. Es wird deutlich, dass der interessante

Bereich ungefähr zwischen null und zwei liegen muss, da für größere Parametrisierungen kontinuierlich immer schlechtere Werte erzielt werden. Auffallend ist, dass das *F-Measure* sich nicht der Genauigkeit der Basis-Heuristik *Recall* (55.4776 %) annähert. Das *Klösgen-Maß* hingegen konvergiert gegen die Heuristik *Coverage*. Zu erkennen ist, dass sich für das *m-Estimate* ein komplett anderes Bild ergibt, da hier die Genauigkeiten stark schwanken und kein monotones Verhalten bei den Ergebnissen zu sehen ist. Daher sollte bei dieser Heuristik ein anderes Intervall getestet werden.

Nach diesem ersten Testlauf kann mit der Suche nach dem besten Parameter begonnen werden, da man nun für die Heuristiken einen Überblick über die zu prüfenden Bereiche gewonnen hat. Diese Suche ist manuell durchgeführt worden. Der folgende Pseudo-Code skizziert die verwendete Prozedur. Je nachdem mit welchen Werten man die sortierte Liste *Parameters* füllt, ergeben sich die verschiedenen Durchläufe. Das eigentliche Suchverfahren entspricht daher den verschiedenen Aufrufen der Prozedur.

Algorithm 2 Die Prozedur zum Finden des besten Parameters in einem vorgegebenen Intervall

```
PROCEDURE GetBestParameter (currentHeuristic, parameters, dataSets)
{
  bestValue = 0
  bestParameter = 0
  # durchsuche alle Parameter
  FOREACH parameter IN parameters
  {
    # schreibe die Genauigkeit des Parameters in die Variable currentValue
    currentValue = GetAccuracy (currentHeuristic, parameter, dataSets)
    # ist die Genauigkeit höher, setze sie als beste Genauigkeit und speichere
    # den Parameter
    IF (currentValue > bestValue)
    {
      bestParameter = parameter
      bestValue = currentValue
    }
  }
# gib den Parameter mit der höchsten Genauigkeit zurück
RETURN (bestParameter)
```

Die Methode *GetAccuracy* bekommt die aktuelle Heuristik und die Liste der Parameter und Datenmengen übergeben. Sie ruft den Covering-Algorithmus auf, der im Abschnitt 4.2 beschrieben wurde. Der Algorithmus bekommt eine Trainingsmenge und eine Heuristik mit ihrem Parameter übergeben. In dieser Prozedur wird dann der Regel-Lerner instantiiert und eine Regelmenge für die aktuelle Datenmenge gelernt. Zurückgegeben wird die Genauigkeit der aktuellen Heuristik mit ihrem Parameter auf der Datenmenge (da der Algorithmus im Rahmen einer Methode zur Evaluation aufgerufen wird). Diese wird mit der Cross-Validation ermittelt, die in Abschnitt 4.3.1 erläutert wurde. Die Methode *GetAccuracy* addiert nun für die aktuelle Heuristik mit ihrem Parameter die Genauigkeiten jeder Datenmenge und teilt diesen Wert dann durch die Anzahl an Datenmengen. Den so

ermittelten Wert, also die durchschnittliche Genauigkeit der Heuristik mit dem Parameter auf allen Datenmengen, gibt die Prozedur dann zurück.

Für die Bestimmung des besten Parameters ist eine weitere Prozedur nötig. Diese funktioniert analog wie *GetBestParameter*, nur dass nicht der beste Parameter, sondern die höchste Genauigkeit zurückgegeben wird. Die Prozedur wird im Folgenden als *GetHighest-Accuracy* bezeichnet.

5.1.1 Intervallschachtelung und das Suchverfahren

Um den besten Parameter zu finden, wäre es optimal, eine vollständige Suche durchzuführen. Hierbei würden alle möglichen Parametrisierungen getestet. Da aber unklar ist, welche Schrittweite man wählen sollte und es unmöglich ist, so viele unterschiedliche Werte zu durchsuchen, muss ein anderes Verfahren verwendet werden. Daher wird im ersten Durchlauf ein größerer Bereich mit einer bestimmten äquidistanten Schrittweite durchsucht. Dieser Bereich wird nach den Kriterien ausgewählt, die sich in der ersten Test-Serie ergeben haben. So wird beim *Klösgen-Maß* und beim *F-Measure* nur das Intervall von null bis eins durchsucht[1]. Beim *m-Estimate* ist man gezwungen deutlich mehr Parametrisierungen zu testen, da man nach dem ersten Suchlauf keine eindeutigen Tendenzen feststellen konnte.

Nach dem ersten Durchlauf kann nun mit der eigentlichen Suche begonnen werden. Die Idee hierbei ist, Parameter zu finden, für die hohe Genauigkeiten erzielt werden. Dann wird ein bestimmter Bereich um diese lokalen Maxima genauer durchsucht, wobei wieder ein bester Parameter gefunden wird. Um diesen herum wird noch genauer gesucht. Dieses Verfahren wird so lange wiederholt, bis sich keine signifikante Verbesserung der Genauigkeit mehr ergibt. Es handelt sich bei dieser Methode um eine Intervallschachtelung. Man beginnt mit einem bestimmten Intervall und greift sich aus diesem einen kleineren Bereich heraus.

Definition 5.1 (Intervall) Ein Intervall ist eine Teilmenge einer Menge von Objekten, die eindeutig geordnet sind. Die Menge, die das Intervall beschreibt ist entweder leer, enthält genau ein Element oder enthält mehr als ein Element und alle Elemente sind geordnet. In den folgenden Abschnitten gilt: $[a, b] = \{x \in \mathbb{R} | a \leq x \leq b\}$.

Aus diesem Intervall wird nun wiederum ein kleinerer Bereich ausgewählt. Führt man diese Schachtelung unendlich oft hintereinander aus, so zieht sich das Intervall immer mehr zusammen, bis es einem einzigen Punkt entspricht. Tendenziell hat sich bei der Suche ergeben, dass nach ungefähr 5 Durchläufen die Genauigkeit nicht mehr weiter verbessert werden konnte.

[1] Beim *Klösgen-Maß* ist bereits ab einem Parameter von 1.1 eine kontinuierliche Verschlechterung erkennbar und Werte unter eins haben deutliche höhere Genauigkeiten.

Durchlauf	zu durchsuchende Menge	Schrittweite	bester Parameter	Genauigkeit
1	$\{0.1, ..., 1.0\}$	0.1	0.4	84,5658
2	$\{0.35, ..., 0.45\}$	0.01	0.42	84,6852
3	$\{0.415, ..., 0.425\}$	0.001	0.418	84,7015
4	$\{0.4175, ..., 0.4185\}$	0.0001	0.4176	84,7045
5	$\{0.41755, ..., 0.41765\}$	0.00001	0.4176	84.7045

Tabelle 5.2: Beispiel für die Suche

Es gibt verschiedene Möglichkeiten wie man die untere Grenze a und die obere Grenze b des Intervalls um den bis dahin besten Parameter p_{best} bestimmt und wie man die Schrittweite s wählt. Je weiter weg sich die Grenzen vom lokalen Maximum befinden, desto höher ist die Wahrscheinlichkeit das globale Optimum zu finden. Man muss sich hier aber einschränken, da man keine unbegrenzte Rechenzeit zur Verfügung hat. Bei dem verwendeten Verfahren wurden jeweils 11 Werte getestet[2]. Die neuen unteren und oberen Grenzen des Intervalls berechnen sich wie folgt.

$$a = p_{best} - \frac{s}{2},\ b = p_{best} + \frac{s}{2} \text{ und } s = \frac{s}{10} \tag{5.1}$$

Hat nach dem ersten Durchlauf zum Beispiel der Parameter 0.4 die höchste Genauigkeit erreicht, so wird im nächsten Testlauf die Menge $\{0.35, 0.36, ..., 0.4, 0.41, ..., 0.45\}$ mit der Schrittweite 0.01 durchsucht. Die Parameter im Bereich $\{0.3, ..., 0.34, 0.46, ..., 0.5\}$ werden mit dieser Methode nicht getestet. Da aber beim vorherigen Testlauf die Parameter 0.3 und 0.5 bereits getestet wurden und eine geringere Genauigkeit aufgewiesen haben als der Wert 0.4, ist die Wahrscheinlichkeit gering, dass sich in diesem Bereich ein besserer Parameter befindet. Damit man sich ein Bild vom Ablauf einer Suche machen kann, ist diese beispielhaft in Tabelle 5.2 dargestellt.

Die Genauigkeiten verändern sich in späteren Durchläufen immer weniger, bis keine Verbesserung mehr feststellbar ist. Bei der Suche hat sich ergeben, dass ungefähr ab der vierten Nachkommastelle entweder die gleiche oder eine geringere Genauigkeit erzielt wird. In Tabelle 5.2 ist die Intervallschachtelung zu erkennen, die den Bereich um ein lokales Maximum so lange weiter einschränkt, bis der optimale Parameter gefunden ist. Der Algorithmus 3 durchsucht das aktuelle Intervall so lange nach dem besten Parameter, bis die Veränderung in der Genauigkeit einen bestimmten Schwellwert unterschreitet.

Bei dem Suchverfahren treten verschiedene Probleme auf. Es kann passieren, dass man den besten Parameter beim Suchen verpasst. Dieser Fall wird in der Grafik 5.2 deutlich.

[2]Fünf Parameter unter dem bisher besten, der beste Parameter selber und fünf über dem besten.

Algorithm 3 Der Suchalgorithmus

```
PROCEDURE SearchBestParameter (lowerBorder, upperBorder, increment, currentHeuristic,
dataSets)
{
  bestParameter = 0
  bestAccuracy = 0
  formerAccuracy = bestAccuracy
  threshold = 0.0001
  # erstelle eine Liste, die die zu prüfenden Parameter enthält
  parameters = CreateList (lowerBorder, upperBorder, increment)
  # finde den besten Parameter in der Parameterliste
  bestParameter = GetBestParameter (currentHeuristic, parameters, dataSets)
  # finde die höchste Genauigkeit heraus
  bestAccuracy = GetHighestAccuracy (currentHeuristic, parameters, dataSets)
  # wenn sich keine nennenswerte Verbesserung ergibt, gebe den besten Parameter
  # zurück und breche ab
  IF (bestAccuracy - formerAccuracy < threshold)
  {
    RETURN (bestParameter)
    BREAK
  }
  # rufe die Prozedur rekursiv mit den neuen Intervallgrenzen auf
  SearchBestParameter (bestParameter - increment/2, bestParameter + increment/2,
  increment/10, currentHeuristic, dataSets)
}
```

Hat man im ersten Testlauf den Bereich zwischen null und zehn gewählt und herausgefunden, dass alle Parameter von zwei bis zehn schlechtere Genauigkeiten aufweisen als der Wert eins, so durchsucht man bei einer Bereichsgröße von 11 in der nächsten Einschränkung die Werte zwischen 0.5 und 1.5 in 0.1er Schritten. Da aber der Parameter 0.4 die höchste Genauigkeit hat, wird man hier den besten Wert nicht finden.

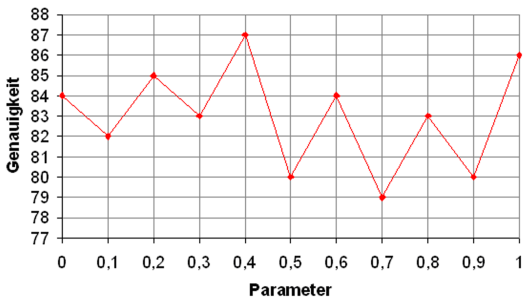

Abbildung 5.2: Schwankende Genauigkeiten

Es tritt aber noch ein anderes Problem auf. Da die Prozedur *GetBestParameter* immer nur einen besten Parameter zurückgibt, werden andere lokale Optima, die während der Suche gefunden wurden, nicht weiter beachtet. Wenn aber eines dieser lokalen Optima bei einer weiteren Einschränkung eine höhere Genauigkeit erzielt, als der zurückgegebene beste Parameter, hat man nicht das globale Maximum gefunden. Durchsucht man für das obige Beispiel den Bereich zwischen 0.1 und 1.0, so treten zwei lokale Optima bei 0.4 und bei 1.0 auf. Hier würde der Algorithmus versuchen ausschließlich den Parameter 0.4 weiter zu verbessern. Erzielt aber eine Einschränkung des Wertes 1.0 eine noch höhere Genauigkeit, so hat man den global besten Parameter nicht gefunden. Als Abhilfe könnte

man den Bereich um jedes lokale Optimum durchsuchen. Dafür muss man definieren, ab wann ein guter Parameter als lokales Optimum gilt. Hierfür gibt es zwei verschiedene Möglichkeiten. Einerseits könnte man die Anzahl an lokalen Optima festlegen und andererseits wäre es möglich, eine Grenze einzuführen, ab der ein Maximum noch als lokales Optimum gilt. Legt man die Anzahl an lokalen Optima auf drei fest, dann werden nur die drei besten Kandidatenparameter weiter verfeinert. Es ist schwierig, eine gute Grenze einzuführen, ab der man noch von einem lokalen Optimum sprechen kann. Außerdem würde die Grenze bei allen drei Heuristiken anders gewählt werden müssen. Aus diesem Grund ist es sinnvoller, die Anzahl der lokalen Optima auf drei zu begrenzen. Dafür muss aber die Prozedur *GetBestParameter* verändert werden. Sie gibt nun nicht mehr einen besten Parameter zurück, sondern eine Liste mit maximal drei Kandidatenparametern[3].

Nach dem ersten Durchlauf liegen die Maxima meistens bei Werten, die nicht getestet wurden, da zwischen den einzelnen Parametern die Genauigkeiten interpoliert werden. In der Grafik 5.3 werden die lokalen Optima durch blaue Punkte repräsentiert. Es lassen sich Parameter identifizieren, die eine hohe Genauigkeit erzielen, aber keine Optima sind. Dieser Sachverhalt wird deutlich, wenn man sich die Kurve des *Klösgen-Maßes* nach dem ersten Durchlauf betrachtet.

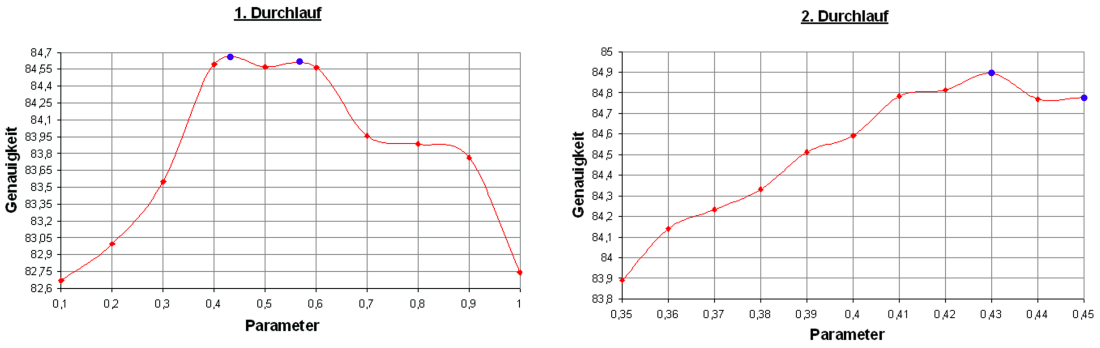

Abbildung 5.3: Die ersten beiden Durchläufe des Klösgen-Maßes

Beim nächsten Testlauf liegen die lokalen Optima, die wieder den blauen Punkten entsprechen, bei getesteten Parametern. In der Kurve zum zweiten Durchlauf muss man allerdings davon ausgehen, dass der Wert 0.45 noch ein weiteres lokales Optimum darstellt. Die Tendenz ist steigend und es ist unbekannt, welche Genauigkeit der nächstgrößere Parameter erreicht. Beim *Klösgen-Maß* und beim *F-Measure* hat die Einschränkungen der anderen Kandidatenparameter keinen neuen global besten Parameter gebracht. Beim *m-Estimate* zeigt sich hingegen ein anderes Verhalten. Hier erzielen auch die lokalen Optima sehr hohe

[3]Es wird nun nicht mehr ein einzelner Wert, sondern eine Liste zurückgegeben. In der Schleife würde ein Hash mit den Parametern als Schlüssel und den korrespondierenden Genauigkeiten als Wert gefüllt werden. Dieser wird absteigend nach der Genauigkeit sortiert und die ersten drei Parameter in die Rückgabeliste gespeichert.

Genauigkeiten und müssen deshalb weiter eingeschränkt werden. Da die Suche immer aufwändiger wird, je häufiger man die lokalen Optima ebenfalls testet, ist die Entscheidung getroffen worden, nur beim *m-Estimate* auf die anderen Kandidatenparameter einzugehen. Es hat sich herausgestellt, dass bereits nach dem ersten Durchlauf eine genauere Prüfung sinnvoll ist. Nach dem zweiten Testlauf wurde dann nur noch ein Parameter weiter verfeinert, da es zu aufwändig gewesen wäre in jedem Schritt auf alle lokalen Optima einzugehen.

5.1.2 m-Estimate

Das *m-Estimate* unterscheidet sich in einigen wichtigen Punkten von den beiden anderen parametrisierbaren Heuristiken. Einerseits ist der Unterschied in der Genauigkeit der beiden Basis-Heuristiken viel geringer. Es ist zwar möglich, dass für gewisse Parametrisierungen auch Werte erreicht werden können, die unterhalb der Genauigkeit von *Precision* liegen, man kann jedoch davon ausgehen, dass diese Basis-Heuristik ungefähr die schlechteste Genauigkeit repräsentiert. Daher können sich die Genauigkeiten nur in einem kleinen Bereich bewegen. Dieser Wertebereich der Genauigkeiten liegt ungefähr zwischen 82% und 86%. Andererseits ist der Unterschied in den Isometriken zwischen den beiden Basis-Heuristiken geringer als beim *Klösgen-Maß* und beim *F-Measure*. Aus beiden Eigenschaften ist abzuleiten, dass das Intervall, in dem sich der optimale Parameter befindet, deutlich größer ist.

Im ersten Durchlauf hat sich herausgestellt, dass es in kleinen Bereichen bis zum Parameter 0.01 zu keiner Änderung in der durchschnittlichen Genauigkeit kommt. Es ergibt sich der Wert von *Precision* mit 82.4956%. Bei ansteigendem Parameter sind große Schwankungen beobachtbar. Da nach dem ersten Durchlauf, in dem der Bereich zwischen 1 und 20 getestet wurde, noch keine aussagekräftigen Tendenzen festgestellt werden konnten, muss die obere Grenze weiter erhöht werden. Letztendlich resultiert ein Intervall von $[1, 90]$. In diesem sollte sich der beste Parameter befinden. In der Grafik 5.4 sind die Genauigkeiten für dieses Intervall in 5er Schritten aufgetragen. Die unbekannten Werte sind interpoliert worden.

Es ist zu erkennen, dass sich durchgehende Schwankungen in der Genauigkeit ergeben. Ab einem Wert von 90 sinkt diese dann dauerhaft unter 85.1%. Das Intervall ist bis zum Parameter 35 in 1er-Schritten durchsucht worden, da ab diesem Wert die Genauigkeit unter 85.3% gefallen ist und später nicht mehr anstieg. Es haben sich drei lokale Maxima ergeben: Die Parameter 14, 23 und 26. Ihre Genauigkeit lag jeweils bei ca. 85.6%. Da der Parameter 14 das beste Ergebnis erzielte, sollte man davon ausgehen, dass um diesen Wert herum das globale Optimum liegen muss.

Im zweiten Durchlauf sind dann die Bereiche um diese drei besten Parameter weiter eingeschränkt worden. Hier hat sich der Bereich um 23 herum als der mit der höchsten Genauigkeit erwiesen. Als bester Wert hat sich nach weiteren Testläufen der Parameter 22.466

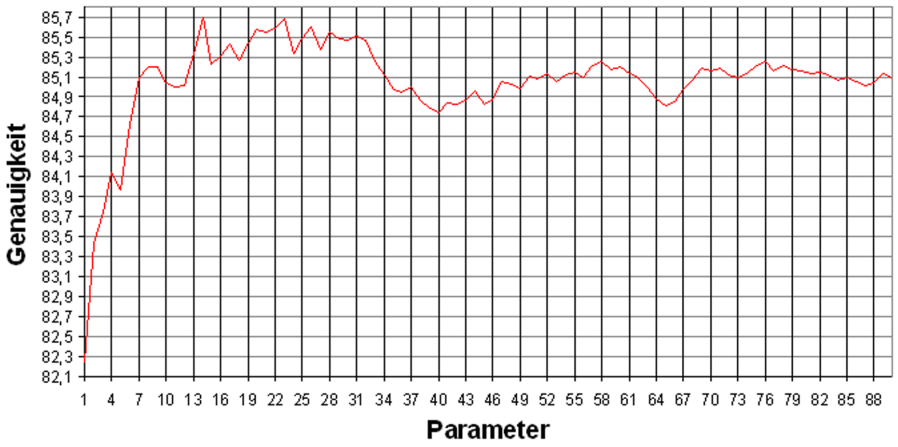

Abbildung 5.4: Kurve für ein großes Intervall

ergeben, der die höchste Genauigkeit von 85.8003% erreicht hat. Da der Kandidatenparameter 23 aber eine geringere Genauigkeit als der um 14 hatte, wäre dieser Wert ohne einen Test der anderen lokalen Optima nicht gefunden worden.

Parameter	Genauigkeit
0.01	82.4956
0.5	82.1344
1.0	82.2405
2.0	83.4355
5.0	83.9729
8.0	85.2046
13.97	85.7638
22.0	85.5825
22.466	85.8003
26.2	85.6940
50.0	85.1082
200.0	84.5329
500.0	83.9561
5000.0	83.0009
1000000.0	82.6665

Tabelle 5.3: Parametrisierungen des *m-Estimates*

Die in der Tabelle 5.3 aufgeführten Parameter sind repräsentativ für die Veränderungen in den Genauigkeiten. Für alle unbekannten Werte wurde die Prozentzahl an korrekt klassifizierten Beispielen interpoliert. Man kann erkennen, dass bei dieser Heuristik die großen Parameter besser abschneiden, als die kleinen. Das liegt an der höheren Genauigkeit der Basis-Heuristik *Weighted Relative Accuracy* im Vergleich zu *Presision*. Außerdem könnte dieser Umstand auch für die Schwankungen bei dieser Heuristik verantwortlich sein. Zu erkennen sind die Unregelmäßigkeiten bei 13.97 und 22.466.

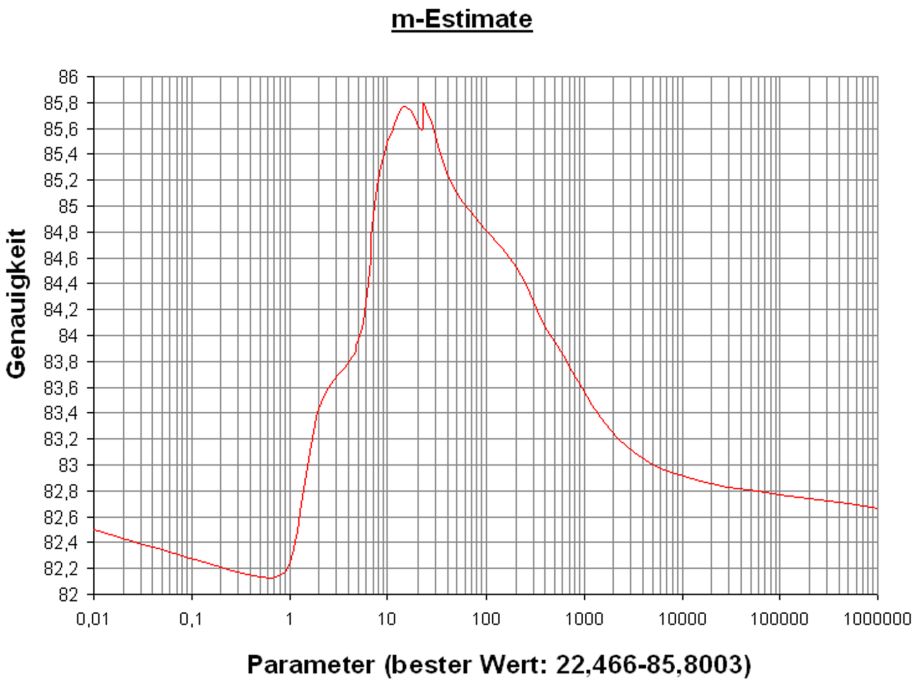

Abbildung 5.5: Kurve für das *m-Estimate*

5.1.3 Klösgen-Maß

Die Suche nach dem besten Parameter beschränkt sich bei dieser Heuristik auf das Intervall $[0, 1]$, welches dem Trade-Off zwischen *Precision* und *Weighted Relative Accuracy* entspricht. Dieser Bereich ist im ersten Durchlauf in 0.1er Schritten durchsucht worden. Die Genauigkeit des *Klösgen-Maßes* ist bis zum Parameter 0.0001 die Gleiche wie die von *Precision*. Von 0.1 bis 0.4 ist dann ein stetiges Ansteigen beobachtbar. Ab einem Wert von 0.6 fällt die Genauigkeit wieder. Bei einem Parameterwert von eins entspricht das *Klösgen-Maß* der Basis-Heuristik *Weighted Relative Accuracy* und erzielt wie zu erwarten den gleichen Wert. Erhöht man den Parameter weiter, fällt die Genauigkeit unter 82%. Nach dem ersten Suchlauf haben sich drei Parameter ergeben, deren Genauigkeit über 84.5% liegt. Diese sind im zweiten Durchlauf weiter eingeschränkt worden. Die besten Parameter nach diesem Testlauf sind 0.57 mit 84.7981%, 0.51 mit 84.7838% und 0.43 mit 84.8944%. Da die Genauigkeit des Kandidatenparameters 0.4 um 0.1% höher war, als die der anderen beiden, wurden diese lokalen Optima nicht weiter eingeschränkt. Als repräsentative Parameter sind die in der Tabelle dargestellten verwendet worden, wobei der Parameter 1.1 hinzu genommen wurde, um aufzuzeigen, dass ab diesem Wert die Genauigkeit stetig fällt.

Parameter	Genauigkeit
0.0001	82.4956
0.1	82.6678
0.2	82.9947
0.3	83.5476
0.4	84.5928
0.4323	84.9909
0.5	84.5702
0.7	83.9554
0.8	83.8842
1.0	82.7405
1.1	79.4810
2.2	66.7201
10.0	57.2531
500.0	55.8597

Tabelle 5.4: Parametrisierungen des *Klösgen-Maßes*

Die Kurve des *Klösgen-Maßes* weist eine starke Ähnlichkeit mit der erwarteten Kurve auf. Auch hier kam es zu kleineren Schwankungen. Es hat sich aber bereits nach dem zweiten Durchlauf ergeben, dass einer der Kandidatenparameter eine deutlich höhere Genauigkeit als die anderen erreicht hat. Daher ist auf die anderen Kandidaten nicht näher eingegangen worden.

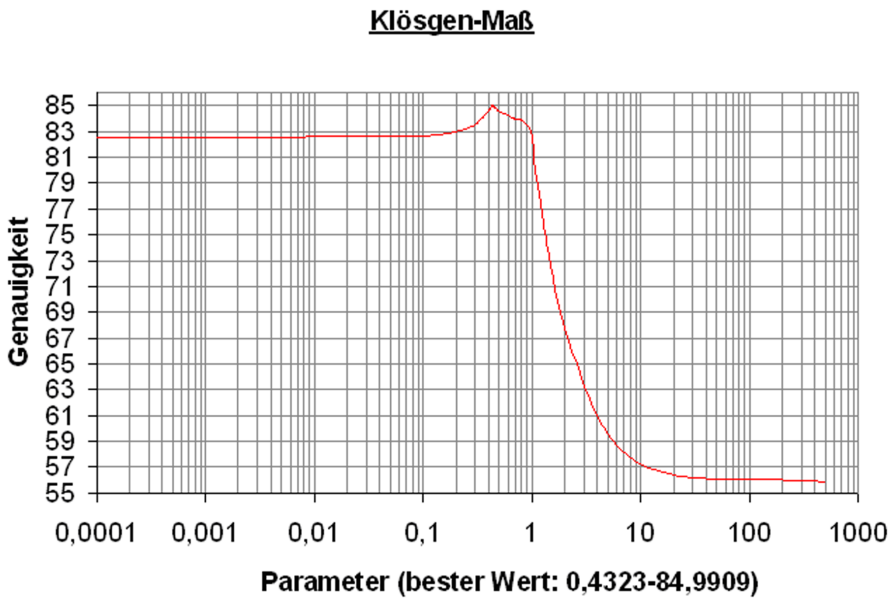

Abbildung 5.6: Kurve für das *Klösgen-Maß*

5.1.4 F-Measure

Auch beim *F-Measure* ist das zu durchsuchende Intervall auf $[0, 1]$ beschränkt, da die Genauigkeit ab einem Parameter von 1 bereits unter 80% fiel. Bei dieser parametrisierbaren Heuristik ist bis zu einem Parameter von 0.01 eine sehr geringe Verschlechterung in der Genauigkeit erkennbar. Sie konvergiert gegen 82.4956%, dem Wert von *Precision*. Schon nach dem ersten Durchlauf hat sich als bester Parameter der Wert 0.5 mit einer Genauigkeit von 84.2904% ergeben, der auch in weiteren Einschränkungen nicht mehr verbessert werden konnte. Als weitere Kandidaten für optimale Werte galten nach dem zweiten Durchlauf ausschließlich Parameter in der Region von 0.5. So erzielte der Wert 0.493 eine Genauigkeit von 84.1025% und der Wert 0.509 erreichte 84.2606%. Beide Parameter ließen sich aber durch weitere Einschränkungen nicht mehr verbessern. Eine kleine Schwankung fiel beim Parameter 0.2 auf, wo die Genauigkeit kurz anstieg, wie in Tabelle 5.5 zu sehen ist. Da aber die Werte kurz nach 0.5 ein deutlich besseres Ergebnis aufgewiesen haben, konnte dieser Ausreißer nicht als lokales Optimum gelten.

Parameter	Genauigkeit
0.01	82.4956
0.1	82.4963
0.2	83.2370
0.3	82.9694
0.4	83.0603
0.5	84.2904
0.6	83.7692
0.7	83.4052
0.8	83.1898
0.9	82.3027
1.0	79.2174
2.5	73.6399
5.0	69.0166
10.0	67.3479
100000000.0	64.5480
1000000000.0	55.4776

Tabelle 5.5: Parametrisierungen für das *F-Measure*

Das *F-Measure* näherte sich ebenfalls an die erwartete Kurve an. Es gab nur wenige Schwankungen und diese waren alle in der Umgebung des besten Parameters. Ein Unterschied zu den beiden anderen Heuristiken bestand darin, dass es bei sehr großen Parametrisierungen nochmals zu einem signifikanten Abfall in der Genauigkeit kam.

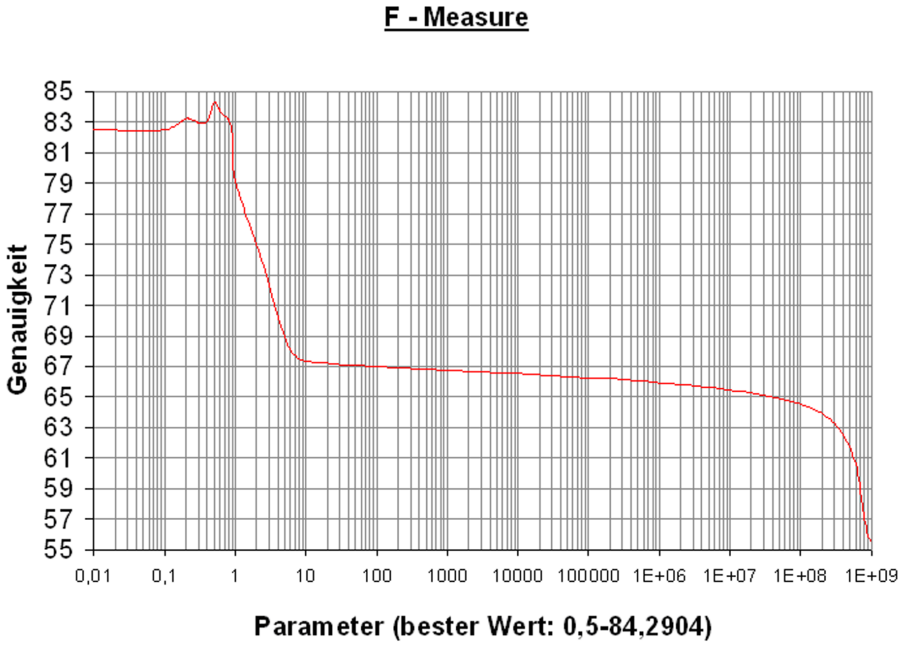

Abbildung 5.7: Kurve für das *F-Measure*

5.2 Vergleich der parametrisierbaren Heuristiken

Das *m-Estimate* mit Parameter 22.466 hat mit Abstand die höchste Genauigkeit von 85.8003 % erreicht. Dieser Wert ist signifikant besser als der der beiden anderen Heuristiken. Das *Klösgen-Maß* mit 0.4323 ist die zweitbeste Heuristik mit 84.9909 % korrekt klassifizierten Beispielen. Damit ist sie um fast einen ganzen Prozentpunkt schlechter als das *m-Estimate*. Die Genauigkeit des *F-Measures* ließ sich leider nicht besonders erhöhen. Hier erreicht bereits einer der Standard-Parameter 84.2904 %, was mit einer genaueren Suche nicht mehr verbessert werden konnte. Nimmt man als Richtwert einer hohen Genauigkeit die beste nicht-parametrisierbare Heuristik *Correlation*, die 83.6573 % erreicht, so wird deutlich, dass eine signifikante Steigerung nur mit den beiden Maßen *m-Estimate* und *Klösgen-Maß* möglich ist. Das wirft die Frage auf, warum ausgerechnet das *F-Measure* so schlecht abschneidet. Zur Klärung dieser Frage sollte man sich die Isometriken der jeweils besten Parametrisierung der Heuristiken betrachten. Erkennbar sind hier zunächst große Ähnlichkeiten. Da die Isometriken einer Heuristik die Präferenzen bei Genauigkeit und Abdeckung wiedergeben, liegt es auf der Hand, dass in den Abbildungen gleiche Tendenzen zu beobachten sind.

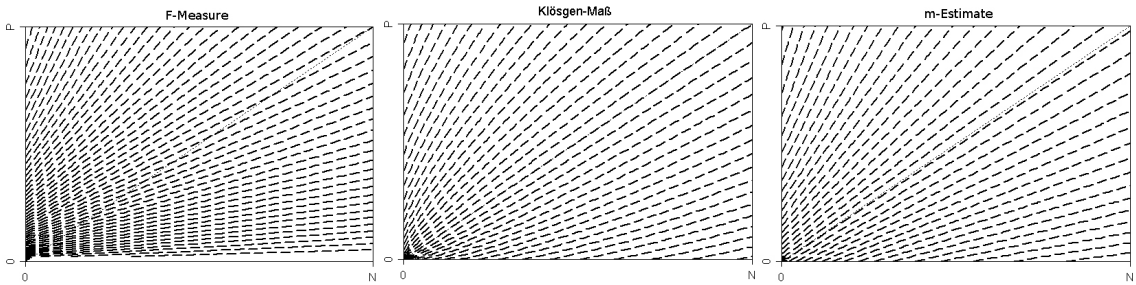

Abbildung 5.8: Isometriken für die besten Parameter

Nur die Isometriken des *F-Measures* zeigen deutliche Unterschiede zu den anderen beiden. Tendenziell gleichen die Isometriken denen der Basis-Heuristik *Precision*. Dieses Verhalten deutet darauf hin, dass die Genauigkeit einer Regel wichtiger als die Abdeckung ist. Beim *m-Estimate* liegt der Ursprung im Punkt $(-n_m, -p_m)$. Die Isometriken wurden für eine Verteilung von 48 positiven und 60 negativen Beispielen erstellt. Daher liegt für diesen Fall der Ursprungspunkt bei $(-12.4811, -9.9849)$. Diesen Isometriken versucht sich auch das *Klösgen-Maß* anzunähern, wobei die Linien hier etwas gekrümmt sind, was beim *m-Estimate* nicht möglich ist. Nur im Bereich einer geringen Abdeckung werden die Regeln unterschiedlich evaluiert. Daher kommt die etwas schlechtere Genauigkeit des *Klösgen-Maßes*. Da beim *F-Measure* der Ursprung nur in eine Richtung verschoben werden kann, ist dieses Maß nicht in der Lage, sich den als optimal geltenden Isometriken des *m-Estimates* anzunähern. Hier liegt der Ursprung beim Punkt $(-0.5, 0)$. Betrachtet man den Bereich wo viele negative und wenig positive Beispiele abgedeckt werden, so kann man erkennen, dass das *F-Measure* den Isometriken von *Precision* gleicht, während die Linien der anderen beiden Maße größere Steigungen aufweisen. Hier liegt genau das Problem des *F-Measures*. Es ist nicht möglich mit einer Parametrisierung des Maßes das gewünschte Verhalten zu erreichen. Für eine gute Bewertung ist dies aber sehr wichtig, wie bei den Isometriken des *m-Estimates* und des *Klösgen-Maßes* zu sehen ist. In Bereichen einer hohen Abdeckung von positiven und einer geringen von negativen Beispielen sind die Linien nahezu äquivalent. Daher wird das *F-Measure* Regeln, die diese Abdeckung haben, gleich gut wie die beiden anderen Heuristiken bewerten, während es in dem anderen Bereich eine wesentlich schlechtere Evaluation als das *Klösgen-Maß* und das *m-Estimate* erreicht. Zu erkennen ist auch, dass Regeln, die eine gleiche Anzahl von positiven und negativen Beispielen abdecken vom *F-Measure* unterschiedlich bewertet werden. So erhält eine Regel, die alle Beispiele abdeckt dieselbe Bewertung wie eine, die eine gewisse Anzahl von positiven und kein negatives Beispiel abdeckt. Dieses Verhalten ist aber für eine Heuristik, die eine hohe Genauigkeit erzielen soll, nicht erwünscht. Selbstverständlich sollte eine Regel, die eine gewisse Anzahl von positiven und keine negativen Beispiele abdeckt, eine höhere Bewertung erhalten als eine Regel, die einfach nur viele Beispiele abdeckt. Generell ist es beim *F-Measure* nur für die beiden Extremwerte möglich symmetrische Isometriken zu erlangen. Da aber sowohl

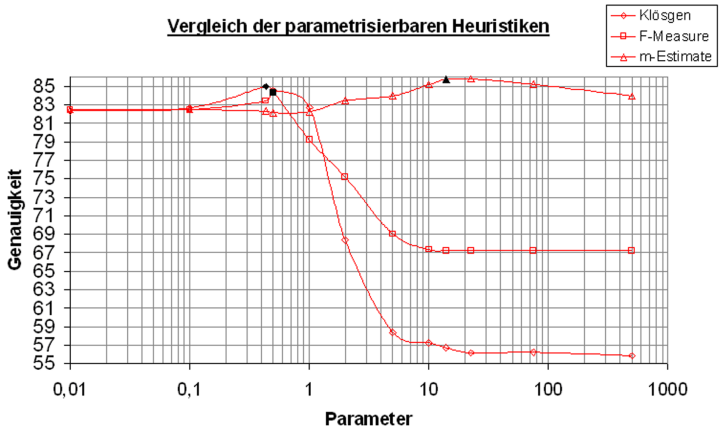

Abbildung 5.9: Vergleich der Heuristiken

die Isometriken vom *Klösgen-Maß* als auch die des *m-Estimates* symmetrisch sind, folgt, dass die Symmetrie für eine gute Heuristik sehr wichtig ist.

Die schwarzen Datenpunkte in der Grafik 5.9 repräsentieren die besten Parameter der einzelnen Heuristiken. Deutlich wird, dass das *m-Estimate* erst bei größeren Parametrisierungen beginnt eine höhere Genauigkeit zu erreichen und bei sehr großen Werten noch äußerst genau ist. Das *F-Measure* fällt dann bei riesigen Parametern nochmal ab und ist dann ebenso gut wie das *Klösgen-Maß*.

Für bessere Vergleichsmöglichkeiten wurde eine große Tabelle erstellt, in der alle Parametrisierungen aus den Tabellen 5.3, 5.4 und 5.5 als eigene Heuristiken angesehen wurden. Es entsteht so eine Gesamtzahl von 45 Heuristiken, die sich aus dem Namen des Maßes und dem Parameter zusammensetzen.

Betrachtet man für jede Datenmenge einzeln die jeweils beste parametrisierbare Heuristik, so lassen sich einige interessante Eigenschaften der Parametrisierungen erkennen.

Alle drei Heuristiken umfassen in etwa gleich viele Datenmengen, bei denen genau ein Parameter den besten Wert erzielt. Interessant sind die Datenmengen „iris“ und „mushroom“, da dort alle Heuristiken mit jeweils unterschiedlichen Parametern die gleiche Genauigkeit erreichen. Auf beiden Datenmengen sind die Parameter gering, woraus folgt, dass alle 3 Heuristiken zur Basis-Heuristik *Precision* tendieren. Das bedeutet, dass diese Basis-Heuristik bereits die höchstmögliche Genauigkeit erreicht hat. An den oberen Grenzen kann man erkennen, ab welchem Parameter die Heuristiken auf diesen Datenmengen zu den jeweiligen anderen Basis-Heuristiken tendieren. Der Vorteil vom *F-Measure* und dem *Klösgen-Maß* liegt darin, dass die beiden Abdeckungsheuristiken mehr auf eine hohe Abdeckung aus sind, als *Weighted Relative Accuracy*.

Datenmenge	Par. Klösgen	Wert	Par. F-Measure	Wert	Par. m-Estimate	Wert
anneal	0.2	99.1228	0.3	99.1228	2, 5, 8	99.3734
audiology	1	85.3982	2.5, 5, 10	85.3982	500, 5000, 1E+6	85.3982
breast-cancer	0.1, 10, 500	70.2797	0.5	74.4755	26.2	73.7762
cleveland-heart-disease	0.7	77.8878	0.6	78.4755	8	79.2079
contact-lenses	0.8, 1, 1.1	83.3333	2.5, 5, 10, 1E+8	83.3333	13.97,..., 1E+6	83.3333
credit	1.1	86.9388	2.5	86.1224	5000, 1E+6	86.7347
glass	0.5	69.1589	1	68.2243	13.97	70.5607
glass2	0.5	82.8221	0.8	80.9816	22.466	84.0491
hepatitis	0.4323	81.9355	0.2	80.6452	8, 13.97	82.5806
horse-colic	1.1	84.5109	0.4	84.2391	22.466	84.5109
hypothyroid	0.7, 0.8	99.2728	2.5	99.2096	500	99.2728
iris	0.0001, 0.1, 0.2, 0.4	95.3333	0.01,...,0.6	95.3333	0.01, 0.5, 1, 2	95.3333
krkp	0.1	99.3742	0.1	99.4681	22, 22.466, 26.2	99.562
labor	0.3	91.2281	0.7	91.2281	8	91.2281
lymphography	0.0001, 0.2, 0.3	82.4324	0.1	83.1081	0.01	82.4324
monk1	0.3	95.9677	0.01, 0.1	94.3548	5, 22	95.9677
monk2	10, 500	62.1302	0.8,...,1E+8	62.1302	2	56.8047
monk3	0.4	88.5246	0.2, 0.4, 0.5	86.0656	8	91.8033
mushroom	0.0001,...,0.4323	100	0.01,...,0.6	100	0.01,...,50	100
sick-euthyroid	0.4	97.3443	0.8	97.1546	200	97.1546
soybean	0.4323	92.8258	0.5	93.3904	22, 22.466	93.1186
tic-tac-toe	0.4	97.3904	0.2	97.3904	50	98.1211
titanic	0.0001,...,0.3	78.328	0.01, 0.1	78.328	0.01,...,26.2	78.328
vote	1.1	95.6322	0.4	95.8621	13.97	95.8621
vote-1	1	89.6552	0.7, 0.8	90.1149	13.97	90.8046
vowel	0.5	74.4444	1	72.0202	50	74.7475
wine	0.8	96.6292	0.8, 2.5, 10	96.0674	200	96.6292
Durchschnitt		87.3297		87.1001		87.6554
Parameter/ Gewinne	0.0001 & 1.1 / 4		2.5 / 6		13.97 / 7	
Genauigkeit des Gewinners	82.2956 & 79.481		73.6399		85.7638	

Tabelle 5.6: Die besten Parameter pro Datenmenge

Das wird bei der Datenmenge „monk2“ deutlich, da hier scheinbar auf Regeln mit hoher Abdeckung abgezielt wird. Die Regelmenge, die für diese Beispielmenge gefunden worden ist, bestand bei beiden parametrisierbaren Heuristiken aus einer einzigen Regel, die der generellsten Regel entspricht und immer *TRUE* ausgibt. Diese Regel deckt alle Beispiele ab, wobei die Trainingsmenge 105 positive Instanzen enthält, woraus sich eine Genauigkeit von 62.1302 % ergibt. Da keine andere Heuristik eine bessere Regel gefunden hat, kann man davon ausgehen, dass diese Regel optimal ist[4]. Es haben insgesamt lediglich

[4]Bei den verwendeten Heuristiken und der Variante des Regel-Lerners ist die Regel optimal.

37 der 52 Heuristiken überhaupt eine höhere Genauigkeit als 50 % (sind also besser als ein Klassifikator, der eine zufällige Vorhersage ausgibt). Eine möglichst allgemeine Regel zu finden, kann mit *Weighted Relative Accuracy* nur schwer erreicht werden[5]. Deshalb hat das *m-Estimate* auf dieser Menge schlechter abgeschnitten. Eine weitere interessante Datenmenge ist „labor". Hier erzielen *Klösgen0.3*, *F-Measure0.7* und *m-Estimate8.0* exakt dieselbe beste Genauigkeit, wobei das *F-Measure* und das *m-Estimate* jeweils die gleichen 3 Regeln gefunden haben und das *Klösgen-Maß* 4 Regeln benötigte. Auf der Trainingsmenge „horse-colic" hat *Weighted Relative Accuracy* die höchste Genauigkeit erzielt. Ebenso gut haben das *m-Estimate* mit dem Parameter $1 * E^6$, das *Klösgen-Maß* mit eins und das *F-Measure* mit 0.4 abgeschnitten. Beim *m-Estimate* zeigt sich hier die Konvergenz gegen diese Basis-Heuristik. Das *Klösgen-Maß* entspricht mit dieser Parametrisierung genau der Heuristik. Auf dieser Datenmenge ist der Parameter 0.4 beim *F-Measure* repräsentativ für *Weighted Relative Accuracy*.

Betrachtet man die Parameter, die am häufigsten zu den besten pro Datenmenge gehörten, so sieht man, dass beim *F-Measure* der Parameter 2.5 mit sechs Siegen am häufigsten gewonnen hat. Dieser Parameter erzielt allerdings lediglich eine durchschnittliche Genauigkeit von 73.6399%. Daher muss er auf den anderen Datenmengen teilweise sehr schlecht abschneiden. Auch beim *m-Estimate* siegt nicht der beste Parameter am häufigsten. Beim *Klösgen-Maß* siegt sogar kein Parameter alleine. Beide Gewinner haben eine signifikant schlechtere Genauigkeit als der optimale Parameter dieser Heuristik.

5.3 Vergleiche mit anderen Heuristiken

Da bisher noch keine Aussage darüber getroffen wurde, wie gut die parametrisierbaren Heuristiken tatsächlich abschneiden, ist nun noch ein großer Vergleich mit anderen Standard-Heuristiken und mit dem Regel-Lerner *JRip* durchgeführt worden. Dazu werden einerseits die 45 einzelnen Parametrisierungen der drei Maße verwendet. Hinzu kommen andererseits noch die 5 Standard-Heuristiken und *JRip* mit und ohne Pruning. Für den Vergleich werden diese 52 Heuristiken miteinander in Beziehung gesetzt. Es werden jeweils die unterschiedlichen Evaluationen Macro-Average-Accuracy, Micro-Average-Accuracy und das Ranking verwendet und in einer großen Tabelle aufgetragen. So kann man die verschiedenen Evaluationsmethoden gut miteinander vergleichen und bekommt einen Überblick über die Güte der einzelnen Parametrisierungen der Heuristiken. Da die Tabelle zu groß ist, um anschaulich dargestellt zu werden, sind nur die besten und die Standard Parameter der Heuristiken absteigend nach der Macro-Average-Accuracy sortiert aufgetragen. Die gesamte Tabelle findet sich im Anhang der Arbeit unter A.1. Beim *F-Measure* kommt noch

[5] Die Heuristik *Coverage* bewertet die universelle Regel bei dieser Datenmenge mit dem Wert $\frac{105+64}{105+64} = 1$. *Recall* ordnet ihr ebenfalls den Wert $\frac{105}{105} = 1$ zu. Nur *Weighted Relative Accuracy* gibt als Evaluationswert $\frac{105}{105} - \frac{64}{64} = 0$ aus.

der Parameter 0.4 hinzu, da dieser auf den Test-Datenmengen (vergleiche Tabelle 5.8) am besten abgeschnitten hat. In Klammern steht die jeweilige Position, die die Heuristik in der Gesamttabelle erhalten hat. Alle Werte der verschiedenen Evaluationen sind in Prozent. *JRip-P* ist die Variante mit abgeschaltetem Pruning. Mit *WRA* wird in der Tabelle die Heuristik *Weighted Relative Accuracy* bezeichnet.

Heuristik	Macro-Average-Accuracy	Micro-Average-Accuracy	Durchschnittsrang	durchschn. # Regeln	durchschn. # Bedingungen
m-Estimate22.466	85.8003 (1)	93.8655 (2)	16.1296 (2)	12.4444	36.8148
Klösgen0.4323	84.9909 (7)	93.6240 (7)	18.6852 (7)	17.0741	46.8889
Klösgen0.5	84.5702 (9)	92.9615 (13)	23.1111 (16)	14.5926	41.0741
JRip	84.4538 (11)	93.7960 (4)	17.3704 (5)	6.8889	16.9259
F-Measure0.5	84.2904 (12)	92.9358 (14)	19.0741 (8)	16.5185	40.7778
JRip-P	83.8836 (17)	93.5507 (9)	21.9259 (13)	13.1852	45.5185
Correlation	83.6573 (19)	92.3941 (24)	25.1481 (25)	13.4444	37.1111
m-Estimate2.0	83.4355 (22)	92.7126 (18)	23.8889 (21)	28.1852	73.2963
F-Measure0.4	83.0603 (24)	92.5296 (21)	25.1111 (24)	19.6296	48.4815
WRA	82.7057 (29)	90.4323 (37)	28.2593 (35)	4.7778	14.4074
Precision	82.4956 (33)	92.2075 (28)	27.8889 (31)	39.9259	99.9259
Laplace	82.2840 (34)	92.2550 (27)	27.2963 (30)	36.4815	91.0370
Accuracy	82.2808 (35)	91.3071 (33)	28.1852 (34)	32.2593	84.0741
F-Measure1.0	79.2174 (39)	87.8226 (41)	28.9815 (37)	9.4074	24.7037

Tabelle 5.7: Verschiedene Evaluationen

Die genaueste Heuristik ist das *m-Estimate* mit dem Parameter 22.466. Auch das *Klösgen-Maß* erreicht mit dem besten Parameter eine höhere Genauigkeit als *JRip*. Nur das *F-Measure* konnte nicht überzeugen. *JRip* braucht mit Abstand am wenigsten Regeln und ist genauer als die Variante ohne Pruning. Das *m-Estimate* benötigt ebenfalls nur ca. 12 Regeln pro Datenmenge, um diese hohe Genauigkeit zu erreichen. Auch das *Klösgen-Maß* kommt mit ca. 17 Regeln aus. Im Vergleich zum Standard-Parameter 0.5 ist gut zu sehen, dass dieser zwar ungefähr vier Regeln weniger benötigt, dafür aber auch um ca. 0.4% schlechter abschneidet. Ist man darauf aus eine möglichst simple Theorie zu schaffen, so sollte man *JRip* benutzen. Bei diesem Regel-Lerner ist es erstaunlich, wie gut er mit so wenig Regeln abschneidet. Die Anzahl an Bedingungen korreliert stark mit den gefundenen Regeln. Generell war mit einem Ansteigen der benötigten Regeln auch immer eine Erhöhung der Bedingungen pro Regel feststellbar. Dies liegt daran, dass immer der gleiche Regel-Lerner verwendet worden ist und nur die Heuristiken ausgetauscht wurden.

Interessant ist die Heuristik *Weighted Relative Accuracy*, da sie mit durchschnittlich 4.7778 Regeln pro Datenmenge mit Abstand am wenigsten Regeln benötigt. Dieser Wert ist er-

staunlicherweise exakt der gleiche, wie der in [29] für nicht geordnete Regeln[6] ermittelte. Bei geordneten Regeln, wo die erste zutreffende feuert, fand *Weighted Relative Accuracy* in [29] im Durchschnitt 4.56 Regeln. Die durchschnittliche Anzahl von Regeln wurde in der Arbeit aus [29] auf 23 Datenmengen des UCI-Repositories mit dem Regel-Lerner *CN2* mit der Heuristik *Weighted Relative Accuracy* experimentell bestimmt. In diesem Paper wurde ebenfalls festgestellt, dass diese Heuristik ca. acht mal kleinere Regelmengen als *Acurracy* erstellt, dafür aber auch um 5 % ungenauer ist. Wie in den Tabellen 5.7 und 5.8 zu sehen ist, war dieser Umstand mit dem hier benutzten Separate-and-Conquer Algorithmus nicht so. Bei diesem erreichte *Weighted Relative Accuracy* sogar eine um ca. 0.5 % höhere Genauigkeit als *Accuracy* auf den 27 Datenmengen. Auf den Test-Datenmengen wurde eine ca. 0.2 % höhere Genauigkeit gemessen. Die kleinen Regelmengen, die *Weighted Relative Accuracy* gefunden hat, deuten darauf hin, dass bei dieser Heuristik eher ein Underfitting, also eine Unterangepasstheit, geschieht. Es wird versucht, die relative Genauigkeit (die äquivalent zu *Precision Gain* aus 3.4.1 ist) zu verbessern, aber trotzdem noch eine hohe Abdeckung zu gewährleisten. Daher kommt es zu eher allgemeinen Regeln, deren Genauigkeit wie in [29] ermittelt wurde, geringer als die von *Accuracy* sein kann, aber nicht muss. Ein weiterer bemerkenswerter Punkt ist die Genauigkeit der Heuristik *Laplace*. Diese ist um ca. 0.2 % schlechter als die von *Precision*. Da aber durch die *a priori*-Abdeckung im Allgemeinen eine höhere Korrektheit erreicht werden soll, kann man davon ausgehen, dass in diesem Fall *Overfitting* stattfindet. Auf den 30 Test-Datenmengen hat sich dieser Umstand dann auch wieder relativiert, da hier *Laplace* ca. 0.6 % besser als *Precision* ist (vergleiche Tabelle 5.8).

Betrachtet man sich nun die verschiedenen Evaluationsmethoden, so fallen deutliche Unterschiede in der Micro-Average-Accuracy auf, da hier z.B. *JRip* den viert besten Wert bekommt, bei der Macro-Average-Accuracy aber nur auf Platz 12 rangiert. Bei dieser Evaluationsmethode ist das *m-Estimate* mit Parameter 26.2 die beste Heuristik. Ansonsten war festzustellen, dass einige Heuristiken mit Micro-Average-Accuracy signifikant anders bewertet wurden als mit der zur Parametersuche verwendeten Macro-Average-Accuracy. Das *Klösgen-Maß* mit Parameter 0.2 und das *F-Measure* mit dem Wert 0.3 beispielsweise wurden mit Micro-Average-Accuracy gleich bewertet und rangierten in der Gesamttabelle auf Platz 18. Mit Macro-Average-Accuracy erhielten sie die deutlich schlechteren Ränge 25 und 26. Auch die Standard-Heuristik *Correlation* wurde anders bewertet. Sie erhielt allerdings mit Micro-Average-Accuracy einen schlechteren Platz. Bei *Weighted Relative Accuracy* zeigte sich ein ähnliches Verhalten. Wie in Abschnitt 4.3.3 aufgezeigt worden ist, bedeutet eine bessere Platzierung mit Micro-Average-Accuracy, dass die Heuristik auf den kleineren Datenmengen größere Ungenauigkeiten aufweist. Ist der Rang mit Macro-Average-Accuracy

[6] Im Fall von nicht geordneten Regeln stehen diese beim CN2-Lerner in einer Liste. Alle Regeln, die auf das Beispiel zutreffen, werden aufsummiert und die Klasse mit den meisten Regeln ausgegeben (Majority-Vote).

besser, so schneidet die Heuristik auf den großen Trainingsmengen schlechter ab. Generell ist die Genauigkeit bei Micro-Average-Accuracy bei allen getesteten Heuristiken größer. Das liegt daran, dass die Heuristiken auf allen großen Datenmengen jeweils sehr hohe Genauigkeiten erzielen. Da aber bei einer Bewertung mit Micro-Average-Accuracy die kleinen Datenmengen bei der Berechnung weniger stark ins Gewicht fallen, erklären sich so die höheren Werte.

5.4 Überprüfung der Allgemeingültigkeit

Die gefundenen optimalen Parameter sollten auf beliebigen Datenmengen gute Ergebnisse erzielen. Es könnte aber sein, dass sie nur auf den Mengen gut abschneiden, auf denen sie bestimmt worden sind. Es ist selbstverständlich, dass die Parametrisierungen an diese 27 Mengen angepasst sind, sie also zu einem gewissen Teil *overfitten*. Herauszufinden wäre, wie stark das *Overfitting* der Heuristiken auf den Datenmengen ist. Man ist natürlich daran interessiert, Parameter zu finden, die im Allgemeinen hohe Genauigkeiten erzielen und nicht nur auf einer Auswahl von Datenmengen gut abschneiden. Daher werden sie in diesem Abschnitt einem Test auf 30 anderen Datenmengen unterzogen. Interessant hierbei ist einerseits wie gut sie auf diesen anderen Mengen abschneiden und andererseits ob sich Tendenzen feststellen lassen, in welchen Parameterbereichen hohe Genauigkeiten erzielt werden. Die parametrisierbaren Heuristiken sind wieder mit den anderen Heuristiken aus den vorherigen Abschnitten verglichen worden.

In Tabelle 5.8 schneidet nun *JRip* mit Pruning am besten ab. Nur 0.3% schlechter ist der beste Parameter vom *m-Estimate*. Sowohl beim *Klösgen-Maß*, als auch beim *F-Measure* erreicht jeweils ein anderer Parameter die höchste Genauigkeit. Bei beiden liegen diese jedoch sehr nahe bei den exemplarisch bestimmten optimalen Parametern. Zieht man die Grenze, ab der ein Parameter als Optimum in Frage kommt bei 78 % Genauigkeit, so ergibt sich unabhängig von den Datenmengen sowohl für das *F-Measure* als auch für das *Klösgen-Maß* ein bester Parameter im Intervall $[0.3, 0.5]$. Beim *m-Estimate* ist der Parameter 22.0 nur ca. 0.1 % schlechter als das Optimum. Wie schon bei der Suche nach dem besten Parameter, erstreckten sich die Parametrisierungen des *m-Estimates* auf ein deutlich größeres Intervall, als die der beiden anderen Heuristiken. Auf der sicheren Seite ist man, wenn man das Intervall $[13, 27]$ durchsucht. Da die gefundenen Parametrisierungen in diesem Intervall liegen, deutet vieles darauf hin, dass Parameter aus diesem Intervall auch auf anderen Datenmengen gut abschneiden und somit allgemeingültig sind. Des weiteren fällt auf, dass das *Klösgen-Maß* und das *F-Measure* beide eher zu einer Genauigkeitsoptimierung tendieren.

Heuristik	Macro-Average-Accuracy	Micro-Average-Accuracy	Durchschnittsrang	durchschn. # Regeln	durchschn. # Bedingungen
JRip	78.9808 (1)	82.4168 (1)	16.6000 (1)	5.5667	12.2000
m-Estimate22.466	78.6766 (2)	81.7166 (3)	17.9667 (3)	14.3333	47.2667
Klösgen0.5	78.5522 (4)	81.5357 (7)	20.5167 (15)	18.1333	51.4667
JRip-P	78.4999 (5)	82.0392 (2)	18.4667 (5)	14.1333	49.8000
Klösgen0.4323	78.4853 (6)	81.3311 (14)	19.8667 (12)	23.1667	62.6667
F-Measure0.4	78.4695 (8)	81.5986 (5)	17.3833 (2)	23.7667	64.9000
F-Measure0.5	78.1411 (12)	81.5199 (9)	18.2667 (4)	19.5000	52.4333
Correlation	77.5660 (22)	80.9063 (21)	24.7000 (26)	15.4333	47.5000
m-Estimate2.0	77.4737 (23)	80.3949 (23)	23.0667 (24)	37.4000	105.2000
Laplace	76.8890 (28)	79.7577 (30)	26.2667 (31)	45.7667	118.8333
Precision	76.2233 (33)	79.5295 (35)	29.8000 (40)	50.7667	129.1667
WRA	75.7992 (37)	79.3486 (37)	27.0333 (34)	3.7667	12.1333
F-Measure1.0	75.7442 (40)	79.9465 (27)	26.7500 (32)	9.7667	26.4667
Accuracy	75.6023 (41)	78.4675 (39)	31.2333 (42)	39.8000	104.7667

Tabelle 5.8: Verschiedene Evaluationen auf den Test-Datenmengen

Es scheint also generell wichtiger zu sein genaue Regeln zu finden als solche, die viele Beispiele abdecken.

In der großen Tabelle der Parametrisierungen der 27 „Trainings-Datenmengen“ (siehe A.1) und der der 30 „Test-Datenmengen“ (siehe A.2) sollten im besten Fall alle Heuristiken dieselbe Position besetzen. Um die Korrelation der beiden Tabellen aus A.1 und A.2 zu bewerten, ist die in Abschnitt 4.3.4.1 beschriebene Spearman-Rank-Correlation durchgeführt worden. Je höher hier der Wert ist, desto stärker korrelieren die Rankings der Heuristiken in den beiden Tabellen und desto eher sind die gefundenen Parameter allgemeingültig.

Ranking	Macro Average Accuracy	Micro Average Accuracy	# Regeln	# Bedingungen
0.910356	0.917357	0.910868	0.994878	0.994878

Tabelle 5.9: Spearman Rank Correlation

Es ist erkennbar, dass es sich durchweg um eine hohe Korrelation handelt. Es spricht also vieles dafür, dass die gefundenen Parameter auf beliebigen Datenmengen ähnliche Ergebnisse erzielen, wie auf den Mengen, auf denen sie bestimmt wurden. Natürlich könnte die Korrelation auch noch höher sein. Da die Rangliste aber insgesamt 52 Heuristiken umfasst, ist der erreichte Wert ziemlich gut. Auch in uninteressanten Parameterbereichen scheinen sich die gleichen Tendenzen wie auf den Ausgangs-Datenmengen abzuzeichnen. Generell sollte der gefundene Parameter allgemeingültiger werden, je mehr Datenmengen zu seiner Bestimmung verwendet werden.

Kapitel 6

Schlussfolgerungen und Aussichten

6.1 Schlussfolgerungen

In der vorliegenden Diplomarbeit ist es gelungen, für die drei parametrisierbaren Heuristiken *m-Estimate*, *Klösgen-Maß* und *F-Measure* optimale Parameter zu finden, die sich auch auf anderen Datenmengen bewähren und eine höhere Genauigkeit aufweisen, als die Regel-Lern-Verfahren, mit denen sie verglichen wurden. Die Probleme bei der Suche nach guten Parametrisierungen sind aufgezeigt worden und es haben sich Lösungsansätze ergeben. So ist herausgefunden worden, dass es beim Finden der Parameter wichtig ist, nicht nur einen lokal besten Wert weiter einzuschränken, sondern mindestens drei Kandidatenparameter näher zu untersuchen. Ein Beispiel für den Fall, dass nicht der bisher beste Parameter das globale Optimum gewesen ist, wurde anhand des *m-Estimates* illustriert. Des weiteren sind Mängel des Suchverfahrens, die mit der Rechenleistung in Zusammenhang stehen, näher beleuchtet worden. So ist man gezwungen, die Suche auf einen kleinen Bereich einzuschränken (zum Beispiel auf 10 Werte um den bisher besten Parameter herum), während es jedoch optimal wäre, einen möglichst großen Bereich zu durchsuchen. Die Rechenzeit hängt von

- der Anzahl der verwendeten Datenmengen,
- der Größe des Bereichs um den besten Parameter herum, der weiter durchsucht wird, und
- der Anzahl der lokalen Optima, die weiter eingeschränkt werden, ab[1].

[1] Je höher man jeweils einen der drei Faktoren wählt, desto besser werden die Parameter und desto länger die Rechenzeit.

Diese drei Faktoren gilt es, möglichst so zu justieren, dass gute Parameter gefunden werden, aber die Rechenzeit nicht zu hoch wird. Die in der Arbeit verwendeten Eckdaten (27 Datenmengen, 10 Werte und 3 Kandidatenparameter) haben sich als ausreichend erwiesen. Die gefundenen Parametrisierungen des *m-Estimates* und des *Klösgen-Maßes* erzielen auf den Datenmengen, auf denen sie gesucht wurden, die besten Ergebnisse von allen Heuristiken. Das *F-Measure* fällt leider etwas heraus, da es nicht gelungen ist, den Trade-Off so zu konfigurieren, dass ähnlich gute Ergebnisse wie bei den anderen beiden Maßen erreicht werden konnten. Mit den Isometriken dieser Heuristik ist aufgezeigt worden, an welcher generellen Eigenschaft es liegt, dass das *F-Measure* so schlecht abschneidet. Für diese Heuristik und das *Klösgen-Maß* konnte das Intervall $[0.3, 0.5]$ gefunden werden, in dem sich der beste Parameter bei einer Vielzahl von Datenmengen befinden sollte. Beim *m-Estimate* hat sich herausgestellt, dass das Intervall größer ist, da der Bereich, in dem sich der beste Parameter befinden könnte, breiter aufgespannt ist, als bei den anderen beiden Heuristiken. Der optimale Wert liegt in $[13, 27]$.

Zusätzlich wurde in dieser Arbeit ein weit gefächerter Überblick über verschiedene Heuristiken gegeben, die teilweise bereits recht gut erforscht sind (z.B. die Standard-Heuristiken), zum anderen Teil aber noch nicht ausreichend gut verstanden sind. Es ist versucht worden, über die Isometriken der Maße einen Zugang zu einem besseren Verständnis der parametrisierbaren Heuristiken zu bekommen. So kann man beispielsweise beim *m-Estimate* in den Isometriken erkennen, dass durch die geringe Änderung der Anordnung der Linien der für die Parametersuche interessante Bereich größer ausfällt, als bei den anderen Heuristiken. Bei diesen unterscheiden sich die Isometriken der beiden Basis-Heuristiken viel stärker. Die Veränderungen der Anordnung der Linien im PN-Raum bei einer kontinuierlichen Erhöhung des Parameters ist exemplarisch in verschiedenen Grafiken dargestellt worden. Man bekommt einen guten Eindruck, welche Auswirkungen eine Parameteränderung auf die Isometriken der Heuristiken hat. Am interessantesten war hier die Änderung von *Weighted Relative Accuracy* zu *Coverage* beim *Klösgen-Maß,* da sich dort die Gestalt der Isometriken am meisten verändert hat.

Das letzte wichtige Ergebnis der Arbeit ist der umfassende Vergleich der verschiedenen Parametrisierungen mit anderen Heuristiken und untereinander. Eine derart umfangreiche empirische Untersuchung der Heuristiken fehlt bisher im Forschungsbereich des Regel-Lernens. Es ergaben sich einige interessante Punkte, die im Folgenden näher beschrieben werden. Die Heuristik *Precision* hat beispielsweise auf den 27 „Trainings-Datenmengen“ besser abgeschnitten als *Laplace*, welche normalerweise eine höhere Genauigkeit erreichen sollte. Dementsprechend war die Genauigkeit dann bei den 30 „Test-Datenmengen“ auch wieder höher als die von *Precision*. Die Ergebnisse bei der Heuristik *Weighted Relative Accuracy* ähnelten stark denen, die in [29] ermittelt wurden. Für die parametrisierbaren Heuristiken sind verschiedenste Parameterbereiche miteinander verglichen worden und in-

teressante Tendenzen konnten festgestellt werden. Es hat sich beispielsweise herausgestellt, dass die verschiedenen Evaluationsmethoden zu völlig unterschiedlichen Ergebnissen führen. Es wurde aufgezeigt, dass eine Suche der besten Parameter mit anderen Bewertungen alternative Resultate bringen würde und dass weitere Untersuchungen in diesem Bereich sinnvoll wären. Insbesondere eine Bewertung mit dem vorgestellten Ranking-Verfahren (siehe 4.3.4) würde interessante neue Parameter liefern.

6.2 Aussichten

Die Ergebnisse zeigen, dass es möglich ist, gute Parametrisierungen für die drei Heuristiken zu finden. Aus Effizienzgründen ist sowohl die Menge der zu durchsuchenden Parameter als auch die Anzahl der Trainingsmengen beschränkt worden. Hier wäre es interessant zu erfahren, ob man mit einer vollständigeren Suche[2] auf den verwendeten Mengen bessere Parameter gefunden hätte oder ob diese bereits optimal bestimmt worden sind. Andererseits könnte man auch die Anzahl an Trainingsmengen erhöhen, da die Wahrscheinlichkeit immer weiter steigt, noch bessere Parameter zu finden. Die wichtigste Eigenschaft eines guten Parameters ist seine Genauigkeit auf beliebigen Datenmengen. Dass man sich immer an die zur Suche verwendeten Datenmengen anpasst, ist unvermeidbar. Daher sollte die Güte der Parameter immer besser werden, je mehr Trainingsmengen man verwendet.

Interessant wäre herauszufinden, welche Parameter auf den „Testmengen“ gefunden worden wären. Auch das Abschneiden dieser Werte auf den „Trainingsmengen“ wäre von höchster Relevanz, da man den direkten Vergleich mit den für diese Mengen optimalen Parametern bereits vorliegen hätte. Hieraus könnte man gut erkennen, in welchen Bereichen sich die Parameter unterscheiden und wie stark ihre Korrelation ist, wenn man sie auf anderen Datenmengen bestimmt. Welche Trainingsmengen man verwendet, ist ebenfalls von Bedeutung. Hier ist es wichtig, dass ein großes Spektrum unterschiedlicher Datenmengen Verwendung findet. Es sollten Mengen vorhanden sein, die besser mit einer Heuristik erklärt werden können, die die Genauigkeit optimiert und andere, auf denen eine Steigerung der Abdeckung bessere Ergebnisse liefert. Misst man die Genauigkeit der Parameter beispielsweise auf einer einzelnen neuen Datenmenge, so gilt nicht zwingend, dass das *m-Estimate* am besten abschneidet. Interessant wäre deshalb, wenn man herausfinden könnte, welche Eigenschaften eine Datenmenge aufzuweisen hat, auf der eine bestimmte Heuristik (bzw. ein bestimmter Bereich einer parametrisierbaren Heuristik) am besten funktioniert. Dieser Ansatz wird im Bereich des *Meta-Lernens* als das *Aufspüren von Meta-Wissen* bezeichnet. In [31] wird er „Meta-Rules matching domains with Algorithm performance“ genannt

[2] In Abschnitt 5.1 ist erläutert worden, dass immer 10 verschiedene Werte um den bisher besten Parameter durchsucht werden. Hier würde man dann z.B. die Anzahl auf 20 vergrößern.

und funktioniert, indem einerseits Meta-Wissen aus den Charakteristiken der Datenmengen (Informationstheoretisch, statistisch, usw.) und andererseits aus den Eigenschaften des Lern-Algorithmus (Typ, Parametersetzungen, Genauigkeitsmessungen, usw.) erstellt wird. Es wird dann versucht mit einem weiteren Algorithmus zu lernen, wie diese beiden Charakteristiken in Zusammenhang stehen. Soll eine neue Datenmenge klassifiziert werden, misst man die Charakteristiken der Menge und ist dann in der Lage einen geeigneten Klassifikator zu verwenden. Ein guter Überblick über die verschiedenen Ansätze und Forschungsrichtungen beim Meta-Lernen wird in [31] und in [30] gegeben.

Ein ganz anderer Ansatz, der in jedem Fall unterschiedliche Ergebnisse bringen wird, wäre eine Bewertung der Güte eines Parameters mit einer der anderen beschriebenen Evaluationsmethoden. Da die Parameter immer mit Macro-Average-Accuracy gefunden worden sind, geht man implizit davon aus, dass diese Genauigkeitsbestimmung für dieses Problem geeignet ist. Es könnte sich allerdings herausstellen, dass Micro-Average-Accuracy oder auch ein Ranking nicht nur andere beste Parameter liefert (wovon ausgegangen werden kann), sondern dass diese Parametrisierungen dann auf den Datenmengen auch mit Macro-Average-Accuracy besser abschneiden. Selbst wenn dieser eher unwahrscheinliche Fall nicht eintritt, wäre es trotz allem interessant, die resultierenden Unterschiede in den Parametern auszuwerten.

Eine ebenfalls viel versprechende Möglichkeit besteht darin, in jedem einzelnen Conquer-Schritt des Algorithmus eine neue Heuristik zu benutzen, beziehungsweise eine neue Justierung des Trade-Offs vorzunehmen. Aufgrund der hohen Berechnungszeit ist dies sicherlich ein in der Realität nicht praktikabler Ansatz, der aber trotzdem höchst interessante Ergebnisse liefern könnte. Durch eine zusätzliche Gewichtung[3] könnte man entweder mehr auf eine hohe Abdeckung oder auf eine große Genauigkeit abzielen. So wäre es möglich, auch in den letzten Iterationsschritten des Algorithmus noch eine hohe Abdeckung zu fordern und so beispielsweise dem *Problem of Small Disjuncts* [15] entgegenzuwirken oder auch *Overfitting* zu vermeiden. Man könnte auch versuchen, direkt in den ersten Schritten der Covering-Schleife nicht nur auf eine hohe Abdeckung zu setzen, sondern auch eine große Genauigkeit fordern. Dies hätte zur Folge, dass schon am Anfang speziellere Regeln gefunden werden würden, deren Abdeckung deutlich kleiner als bei normalen Separate-and-Conquer Algorithmen wäre. Offensichtlich ist, dass sich der Algorithmus bei dieser Methode noch stärker an die Trainingsmenge anpasst. Bei dem skizzierten Algorithmus würde man pro Trainingsmenge eine bestimmte Anzahl von unterschiedlichen Parametrisierungen erhalten (für jeden Iterationsschritt einen Parameter). Man könnte nun beispielsweise den Algorithmus auf allen Datenmengen laufen lassen und sämtliche Parametrisierungen in einer Liste speichern. Als Auswahlkriterium käme nun eine einfache Mehrheitswahl pro Iterationsschritt in Frage. Man könnte auch für jede Trainingsmenge ein Ranking erstellen

[3] Mit Gewichtung ist eine Einschränkung auf einen bestimmten Parameterbereich gemeint.

und nur die besten drei Heuristiken der endgültigen Liste hinzufügen. Zur Verbesserung der Laufzeit kommen verschiedene Möglichkeiten in Betracht. Einerseits könnte man einen Parameterbereich vorgeben, in dem gesucht wird. Andererseits wäre eine Entscheidung zwischen beispielsweise drei im Vorhinein festgelegten Parametern der Performanz zuträglich. Des weiteren könnte man in den letzten Conquer-Schritten, in denen nur noch wenige, sehr spezielle Beispiele vorhanden sind, auf eine Heuristik setzen, die unabhängig von der Abdeckung der positiven Beispiele nur versucht, negative Beispiele auszuschließen *(MinimizeNegatives)*. Über einen Wert könnte justiert werden, ab welchem Schritt dann diese Heuristik generisch eingesetzt wird.

Die oben skizzierten Ansätze zur besseren Bestimmung der Parameter und zur Optimierung von Separate-and-Conquer Algorithmen an sich sind nur ein kleiner Ausschnitt aus den Möglichkeiten, die es hier noch zu erforschen gibt. Die parametrisierbaren Heuristiken sind noch immer nicht in allen Einzelheiten erklärt. Was beispielsweise passiert, wenn man negative Parameterwerte zulässt, ist unbekannt. Man sieht, dass es in diesem Bereich noch viel zu tun gibt, bis man wirklich alles Potential von Separate-and-Conquer Algorithmen mit parametrisierbaren Heuristiken ausgeschöpft hat.

Literaturverzeichnis

[1] C.A. Brunk and M.J. Pazzani. An Investigation of Noise-Tolerant Relational Concept Learning Algorithms. In *Proceedings of the 8th International Workshop On Machine Learning (ML-91)*, pages 389–393. Morgan Kaufmann: Evaston, Illinois, 1991.

[2] Peter Clark and Tim Niblett. The CN2 induction Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[3] William W. Cohen. Fast Effective Rule Induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[4] T. Fawcett. ROC Graphs: Notes and Practical Considerations for Data Mining Researchers. Technical report hpl-2003-4, HP Labs, 2003. Total Pages = 27.

[5] Peter Flach. Tutorial on: "The Many Faces of ROC Analysis in Machine Learning", July 2004. ICML-04 Tutorial. Notes available from: http://www.cs.bris.ac.uk/~flach/ICML04tutorial/index.html.

[6] Peter A. Flach. The Geometry of ROC Space: Understanding Machine Learning Metrics through ROC isometrics. In *Proceedings 20th International Conference on Machine Learning (ICML'03)*, pages 194–201. AAAI Press, January 2003.

[7] Johannes Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999.

[8] Johannes Fürnkranz. Modeling Rule Precision. Technical Report OEFAI-TR-2003-35, Research Institute for Artificial Intelligence, Wien, Austria, 2003.

[9] Johannes Fürnkranz and Peter A. Flach. An Analysis of Rule Evaluation Metrics. In *Proceedings 20th International Conference on Machine Learning (ICML'03)*, pages 202–209. AAAI Press, January 2003.

[10] Johannes Fürnkranz and Peter A. Flach. An Analysis of Rule Learning Heuristics. Technical Report CSTR-03-002, Department of Computer Science, University of Bristol, February 2003. Pages 0-20.

[11] Johannes Fürnkranz and Peter A. Flach. Roc 'n' Rule Learning - Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(1):39–77, January 2005.

[12] Johannes Fürnkranz and Gerhard Widmer. Incremental Reduced Error Pruning. In *Proceedings the Eleventh International Conference on Machine Learning*, pages 70–77, New Brunswick, NJ, 1994.

[13] Johannes Füurnkranz. FOSSIL: A Robust Relational Learner. *Lecture Notes in Computer Science*, 784:122–137, 1994.

[14] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *EEE Trans. on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[15] Robert Holte, Liane Acker, and Bruce Porter. Concept Learning and the Problem of Small Disjuncts. Technical report, Austin, TX, USA, 1989. Pages 813-818.

[16] Willi Klösgen. Problems for Knowledge Discovery in Databases and their Treatment in the Statistics Interpreter Explora. *International J. Of Intelligent Systems*, 7:649–673, 1992.

[17] Willi Klösgen. Explora: A Multipattern and Multistrategy Discovery Assistant. In *Advances in Knowledge Discovery and Data Mining*, pages 249–271. 1996.

[18] Ron Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, pages 1137–1145, 1995.

[19] Ryszard S. Michalski. On the Quasi-Minimal Solution of the Covering Problem. In *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, volume A3 (Switching Circuits), pages 125–128, Bled, Yugoslavia, 1969.

[20] D.J. Newman, C.L. Blake, S. Hettich, and C.J. Merz. UCI Repository of Machine Learning databases, 1998.

[21] G. Pagallo and D. Haussler. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5:71–99, 1990.

[22] J.R. Quinlan. Learning First-Order Definitions of Functions. *Journal of Artificial Intelligence Research*, 5:139–161, 1996.

[23] J.R. Quinlan and R.M. Cameron-Jones. Induction of Logic Programs: FOIL and Related Systems. *NewGenerationComputing*, 13 (3,4):287–312, 1995.

[24] R.L. Rivest. Learning Decision Lists. *Machine Learning*, 2:229–246, 1987.

[25] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

[26] Steven L. Salzberg. Book review: *C4.5: Programs for Machine Learning* by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, 16(3):235–240, 1994.

[27] Richard Segal and Oren Etzioni. Learning Decision Lists Using Homogeneous Rules. In *National Conference on Artificial Intelligence*, pages 619–625, 1994.

[28] Matthias Thiel. Separate and Conquer Framework und disjunktive Regeln. Master's thesis, TU Darmstadt, 2005.

[29] Ljupco Todorovski, Peter Flach, and Nada Lavrac. Predictive performance of weighted relative accuracy. In Djamel A. Zighed, Jan Komorowski, and Jan Zytkow, editors, *4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000)*, pages 255–264. Springer-Verlag, September 2000.

[30] Ricardo Vilalta and Youssef Drissi. Research Directions in Meta-Learning, August 22 2001.

[31] Ricardo Vilalta and Youssef Drissi. A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Rev*, 18(2):77–95, 2002.

[32] C.S. Wallace and D.M. Boulton. An Information Measure for Classification. *Computer Journal*, 11:185–194, 1968.

[33] I.H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java implementations. In *Proceedings ICONIP/ANZIIS/ANNES'99 Int. Workshop: Emerging Knowledge Engineering and Connectionist-Based Info*, pages 192–196, 1999.

[34] Stefan Wrobel. An Algorithm for Multi-relational discovery of Subgroups. In Jan Komorowski and Jan Zytkow, editors, *Proc. First European Symposion on Principles of Data Mining and Knowledge Discovery (PKDD-97)*, pages 78–87, Berlin, 1997. Springer Verlag.

Anhang A

Tabellen

Die gesamte Tabelle der verschiedenen Evaluationsmethoden für alle Heuristiken auf den 27 „Trainings-Datenmengen“[1]

Heuristik und Parameter	durchschnittlicher Rang	tatsächlicher Rang	Macro Average Accuracy	Rang	Micro Average Accuracy	Rang	Anzahl an Regeln	Rang	Anzahl an Bedingungen	Rang
m-Estimate 22.466	16.1296	2	85.8003	1	93.8655	2	12.4444	24	36.8148	24
m-Estimate 13.97	16.2963	3	85.7638	2	93.8253	3	14.1111	27	39.4444	28
m-Estimate 26.2	15.5370	1	85.6940	3	93.8802	1	11.8148	22	34.1852	22
m-Estimate 22.0	17.5370	6	85.5825	4	93.7667	6	12.2963	23	35.7778	23
m-Estimate 8.0	16.6296	4	85.2046	5	93.5837	8	17.6667	33	48.8889	34
m-Estimate 50.0	19.9815	11	85.1082	6	93.7813	5	9.2593	18	28.1111	21
Klösgen-Maß 0.4323	18.6852	7	84.9909	7	93.6240	7	17.0741	32	46.8889	32
Klösgen-Maß 0.4	19.4444	10	84.5928	8	93.3677	10	18.4444	34	49.7407	35
Klösgen-Maß 0.5	23.1111	16	84.5702	9	92.9615	13	14.5926	28	41.0741	30
m-Estimate 200.0	20.6667	12	84.5329	10	93.0530	12	6.2963	14	19.4444	15
JRip	17.3704	5	84.4538	11	93.7960	4	6.8889	16	16.9259	13
F-Measure 0.5	19.0741	8	84.2904	12	92.9358	14	16.5185	31	40.7778	29
m-Estimate 5.0	19.1481	9	83.9729	13	93.3311	11	21.0741	36	56.8148	36
m-Estimate 500.0	23.8148	19	83.9561	14	92.1050	31	5.7407	13	17.7407	14
Klösgen-Maß 0.7	24.3519	22	83.9554	15	92.4124	23	8.6667	17	25.5185	20
Klösgen-Maß 0.8	23.8704	20	83.8842	16	91.5486	32	6.4074	15	19.4815	16
JRip -P	21.9259	13	83.8836	17	93.5507	9	13.1852	25	45.5185	31
F-Measure 0.6	23.0000	15	83.7692	18	92.8443	16	15.3333	30	38.3704	27
Correlation	25.1481	25	83.6573	19	92.3941	24	13.4444	26	37.1111	25

[1] Beide Tabellen sind nach der Macro-Average-Accuracy geordnet.

F-Measure 0.7	24.3889	23	83.6190	20	92.6540	19	15.1481	29	38.0370	26
Klösgen-Maß 0.3	22.7963	14	83.5476	21	92.9249	15	24.7407	38	63.3333	38
m-Estimate 2.0	23.8889	21	83.4355	22	92.7126	18	28.1852	39	73.2963	39
F-Measure 0.2	23.2593	17	83.2370	23	92.6467	20	30.0370	40	77.0000	40
F-Measure 0.4	25.1111	24	83.0603	24	92.5296	21	19.6296	35	48.4815	33
m-Estimate 5000.0	25.7407	27	83.0009	25	91.0142	34	5.0000	12	15.1481	12
Klösgen-Maß 0.2	23.2963	18	82.9947	26	92.7235	17	30.7407	41	77.6667	41
F-Measure 0.3	26.8704	28	82.9694	27	92.7235	17	24.6296	37	62.6296	37
Klösgen-Maß 1.0	28.1111	33	82.7405	28	90.4359	36	4.8148	11	14.5185	11
Weighted Relative Accuracy	28.2593	35	82.7057	29	90.4323	37	4.7778	10	14.4074	10
Klösgen-Maß 0.1	25.3333	26	82.6678	30	92.4490	22	35.6667	44	89.2593	44
m-Estimate $1 * E^6$	27.9259	32	82.6665	31	90.4432	35	4.8148	11	14.5185	11
F-Measure 0.1	27.1296	29	82.4963	32	92.2990	26	36.1852	45	91.9630	46
F-Measure 0.01	27.8889	31	82.4956	33	92.2075	28	39.8889	48	99.8148	48
Klösgen-Maß 0.0001	27.8889	31	82.4956	33	92.1416	30	39.8889	48	100.0741	51
m-Estimate 0.01	27.8889	31	82.4956	33	92.2075	28	39.9259	49	99.9259	49
Precision	27.8889	31	82.4956	33	92.2075	28	39.9259	49	99.9259	50
Laplace	27.2963	30	82.2840	34	92.2550	27	36.4815	46	91.0370	45
Accuracy	28.1852	34	82.2808	35	91.3071	33	32.2593	42	84.0741	42
m-Estimate 1.0	28.4259	36	82.2405	36	92.3026	25	34.0370	43	86.3333	43
m-Estimate 0.5	30.2037	38	82.1344	37	92.1855	29	37.9630	47	96.0370	47
Klösgen-Maß 1.1	30.3333	39	79.4810	38	88.0092	38	2.2963	7	8.5185	6
F-Measure 1.0	28.9815	37	79.2174	39	87.8226	41	9.4074	20	24.7037	17
F-Measure 0.9	24.3889	23	79.1521	40	87.8409	40	9.3704	19	25.1481	19
F-Measure 0.8	23.8704	20	79.1127	41	87.8518	39	9.4444	21	24.8889	18
F-Measure 2.5	33.7963	40	73.6399	42	75.4584	43	4.3333	9	12.7037	9
F-Measure 5.0	38.5556	41	69.0166	43	69.2032	44	2.8889	8	10.1481	7
F-Measure 10.0	39.6667	42	67.3479	44	67.2706	46	2.2593	6	6.1852	5
Klösgen-Maß 2.2	39.9815	43	66.7201	45	76.9298	42	0.7037	4	0.8519	4
F-Measure $1 * E^8$	42.5000	44	64.5480	46	64.7451	47	1.8519	5	11.6667	8
Klösgen-Maß 10.0	44.9074	45	57.2531	47	67.4499	45	0.3704	3	0.3704	3
Klösgen-Maß 500.0	45.7407	46	55.8597	48	62.0109	48	0.2222	2	0.2222	2
F-Measure $1 * E^9$	45.7778	47	55.4776	49	61.8279	49	0.0000[2]	1	0.0000	1
Durchschnitt F-Measure	29.6412		77.0906		83.9288		14.8079		38.2824	

[2] 0 Regeln bedeutet hier, dass immer die generellste Regel mit dem Kopf *TRUE* gefunden wurde.

Durchschnitt Klösgen-Maß	28.4179		78.2681		87.0706		14.6164		38.3942	
Durchschnitt m-Estimate	21.9876		84.1059		92.8038		17.3753		46.8320	
Durchschnitt JRip	19.6482		84.1687		93.6734		10.0371		31.2222	
Durchschnitt Standard-Heuristiken	27.3556		82.6847		91.7192		25.3778		65.3111	

Tabelle A.1

Die gesamte Tabelle der verschiedenen Evaluationsmethoden für alle Heuristiken auf den 30 „Test-Datenmengen“

Heuristik und Parameter	durchschnittlicher Rang	tatsächlicher Rang	Macro Average Accuracy	Rang	Micro Average Accuracy	Rang	Anzahl an Regeln	Rang	Anzahl an Bedingungen	Rang
JRip	16.6000	1	78.9808	1	82.4168	1	5.5667	15	12.2000	11
m-Estimate 22.466	17.9667	3	78.6766	2	81.7166	3	14.3333	24	47.2667	23
m-Estimate 22.0	18.8000	6	78.5706	3	81.6930	4	14.1000	23	47.5000	24
Klösgen-Maß 0.5	20.5167	15	78.5522	4	81.5357	7	18.1333	29	51.4667	28
JRip -P	18.4667	5	78.4999	5	82.0392	2	14.1333	24	49.8000	26
Klösgen-Maß 0.4323	19.8667	12	78.4853	6	81.3311	14	23.1667	32	62.6667	31
m-Estimate 8.0	19.3500	10	78.4766	7	81.5278	8	22.2000	31	64.3000	32
F-Measure 0.4	17.3833	2	78.4695	8	81.5986	5	23.7667	33	64.9000	33
m-Estimate 13.97	19.1500	9	78.4590	9	81.5435	6	16.5667	26	51.5000	29
m-Estimate 26.2	18.9333	7	78.3411	10	81.4806	11	13.4667	22	44.4667	22
Klösgen-Maß 0.3	19.1000	8	78.1716	11	81.0322	18	33.9667	37	98.0667	37
F-Measure 0.5	18.2667	4	78.1411	12	81.5199	9	19.5000	30	52.4333	30
Klösgen-Maß 0.4	20.4333	14	78.1156	13	81.3941	12	25.4000	34	74.6333	34
m-Estimate 5.0	20.2667	13	78.1109	14	81.1581	16	27.4667	35	74.9667	35
F-Measure 0.3	19.8167	11	78.0917	15	81.1423	17	28.0667	36	76.0667	36
Klösgen-Maß 0.7	20.7667	16	77.9790	16	81.5042	10	8.5667	17	23.2667	17
m-Estimate 50.0	22.8833	23	77.9258	17	81.1423	17	9.0667	20	30.2000	21
F-Measure 0.6	21.1500	17	77.8955	18	81.3626	13	17.7333	28	49.8667	27
F-Measure 0.7	22.8667	22	77.8109	19	81.1974	15	17.2667	27	48.5333	25
Klösgen-Maß 0.8	21.3833	18	77.7921	20	81.1423	17	5.8000	16	17.9000	16
Klösgen-Maß 0.2	21.9667	20	77.6745	21	80.6467	22	39.5000	40	109.2000	41
Correlation	24.7000	26	77.5660	22	80.9063	21	15.4333	25	47.5000	24
m-Estimate 2.0	23.0667	24	77.4737	23	80.3949	23	37.4000	39	105.2000	40
m-Estimate 200.0	22.0667	21	77.2138	24	80.9142	20	5.1000	14	16.7000	15
m-Estimate 500.0	21.8000	19	77.0421	25	81.0086	19	4.6333	13	15.6333	14
F-Measure 0.2	25.9833	29	76.9905	26	80.2140	24	35.7000	38	103.7667	38
Klösgen-Maß 0.1	25.4833	27	76.9052	27	80.1825	25	45.7667	44	120.3333	45
Laplace	26.2667	31	76.8890	28	79.7577	30	45.7667	44	118.8333	44
m-Estimate 1.0	27.4833	36	76.7953	29	79.8600	29	42.6667	42	114.1333	42
F-Measure 0.1	27.9333	37	76.6463	30	79.9780	26	43.7000	43	116.3333	43
m-Estimate 0.5	28.8833	38	76.5666	31	79.7420	31	46.6667	45	122.0000	46

F-Measure 0.01	29.7000	39	76.2248	32	79.5374	34	50.6667	46	128.9667	47
m-Estimate 0.01	29.7000	39	76.2248	32	79.5374	34	50.7667	47	129.1667	48
Precision	29.8000	40	76.2233	33	79.5295	35	50.7667	47	129.1667	48
Klösgen-Maß 0.0001	29.8000	40	76.0871	34	79.3722	36	50.9333	48	129.8333	49
m-Estimate 5000.0	26.1333	30	75.9749	35	79.6161	33	3.9333	12	12.8000	13
m-Estimate $1 * E^6$	26.8167	33	75.8385	36	79.3722	36	3.8333	11	12.2667	12
Weighted Relative Accuracy	27.0333	34	75.7992	37	79.3486	37	3.7667	10	12.1333	10
Klösgen-Maß 1.0	27.1500	35	75.7897	38	79.3407	38	3.7667	10	12.1333	10
F-Measure 0.8	23.6333	25	75.7520	39	79.8914	28	8.9667	19	25.1667	18
F-Measure 1.0	26.7500	32	75.7442	40	79.9465	27	9.7667	21	26.4667	20
Accuracy	31.2333	42	75.6023	41	78.4675	39	39.8000	41	104.7667	39
F-Measure 0.9	25.8833	28	75.4693	42	79.7026	32	8.9333	18	25.3000	19
Klösgen-Maß 1.1	29.8333	41	74.1473	43	77.9876	40	2.5333	9	8.7667	9
F-Measure 2.5	39.0667	44	65.2629	44	71.5365	41	2.5000	8	7.9667	7
Klösgen-Maß 2.2	38.5667	43	62.8008	45	57.1710	45	0.8000	4	1.0667	4
F-Measure 5.0	42.0833	45	61.7648	46	65.2820	42	1.4333	7	5.1000	6
F-Measure 10.0	43.2000	46	60.0045	47	62.6308	43	1.1000	6	4.1333	5
F-Measure $1 * E^8$	44.1500	47	57.9407	48	57.3834	44	0.8333	5	8.6000	8
Klösgen-Maß 500.0	45.3000	48	55.6604	49	51.1447	46	0.2667	2	0.2667	2
Klösgen-Maß 10.0	45.6167	49	55.4188	50	50.9873	47	0.3000	3	0.3000	3
F-Measure $1 * E^9$	46.1000	50	53.2139	51	48.6508	48	0.0000	1	0.0000	1
Durchschnitt F-Measure	29.6229		71.5890		74.4734		16.8708		46.4750	
Durchschnitt Klösgen-Maß	27.5560		73.1128		74.6266		18.4929		50.7072	
Durchschnitt m-Estimate	22.8867		77.4460		80.7138		20.8133		59.2067	
Durchschnitt JRip	17.5334		78.7404		82.2280		9.8500		31.0000	
Durchschnitt Standard-Heuristiken	27.8067		76.4160		79.6019		31.1067		82.4800	

Tabelle A.2