Vergleich von Pruningalgorithmen für Regellerner

Diplomarbeit

eingereicht von

Benedict Werling

am

01. Juni 2008

Betreuer: Prof. Dr. Johannes Fürnkranz
Dipl.-Inf. Frederik Janssen
Eidesstattliche Erklärung

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, 01. Juni 2008
Benedict Werling
Inhaltsverzeichnis

Kapitel 1: Einleitung ............................................................................................................................... 8
  1.1 Motivation und Ziel ....................................................................................................................... 8
  1.2 Überblick über die Kapitel ........................................................................................................ 9
Kapitel 2: Separate-and-Conquer Regellernen ..................................................................................... 10
  2.1 Induktives Lernen ..................................................................................................................... 10
  2.2 Merkmale von Separate-and-Conquer Algorithmen .................................................................. 14
    2.2.1 Hypothesensprache ............................................................................................................. 14
    2.2.2 Suchverfahren .................................................................................................................... 16
    2.2.3 Methoden zu Vermeidung von Overfitting ........................................................................ 18
  2.3 Ein generischer Separate-and-Conquer Algorithmus ................................................................. 19
  2.4 Vergleich von Separate-and-Conquer Algorithmen .................................................................... 21
    2.4.1 Win-Tie-Loss Tabellen .......................................................................................................... 21
Kapitel 3: Suchheuristiken ..................................................................................................................... 23
  3.1 Eigenschaften von Regeln, Regelmengen und Trainingsmengen ............................................... 23
  3.2 PN – und ROC – Raum ............................................................................................................. 24
  3.3 Verwendete Heuristiken ............................................................................................................ 27
    3.3.1 Lineare Heuristiken .............................................................................................................. 27
    3.3.2 Nicht-Lineare Heuristiken .................................................................................................... 33
Kapitel 4: Pruning .................................................................................................................................. 35
  4.1 Postpruning .................................................................................................................................. 37
    4.1.1 REP ....................................................................................................................................... 37
  4.2 Prepruning .................................................................................................................................. 39
    4.2.1 Stopkriterien für das Prepruning .......................................................................................... 39
  4.3 Combining Post- & Prepruning ................................................................................................... 42
  4.4 Integrating Post- & Prepruning .................................................................................................. 43
    4.4.1 IREP – Incremental Reduced Error Pruning ......................................................................... 43
    4.4.2 RIPPER und JRIP ................................................................................................................ 44
Kapitel 5: Technische Umsetzung im SeCo-Framework ........................................................................ 46
  5.1 Das SeCo-Framework ................................................................................................................... 46
    5.1.1 Ein generischer Pruningalgorithmus ...................................................................................... 47
  5.2 Realisierung des Prunings ........................................................................................................... 48
    5.2.1 Pruningklassen ....................................................................................................................... 49
    5.2.2 Pruningoperatoren ................................................................................................................ 49
C7 – Paket „seco.pruning.model“ ................................................................. 175
C8 – Paket „seco.pruning.operator“ ............................................................ 184
Abbildungsverzeichnis

Abbildung 1: Visualisierte Trainingsmenge ................................................................. 13
Abbildung 2: Überangepasste Theorie ....................................................................... 13
Abbildung 3: Geprüfte Hypothese ............................................................................. 13
Abbildung 4: Ein generischer Separate-and-Conquer Algorithmus [Fürn99] .............. 19
Abbildung 5: PN-Raum Beispiel .............................................................................. 25
Abbildung 6: Beispiel Isometrien im PN-Raum ....................................................... 26
Abbildung 7: Verschachtelter PN-Raum (Accuracy Isometrien) ............................... 26
Abbildung 8: MaximizePositives und MinimizeNegatives Isometrien ...................... 28
Abbildung 9: Isometrien von Accuracy und Weighted Relative Accuracy ................. 29
Abbildung 10: Isometrien Precision und Laplace ..................................................... 29
Abbildung 11: Isometrien für das $m$-Estimate mit verschiedenen Parametern .......... 31
Abbildung 12: Isometrien des Klösgen-Maßes für verschiedene Werte von $\omega$ ....... 32
Abbildung 13: Isometrien für Correlation .................................................................. 33
Abbildung 14: Isometrien FOILGain für verschieden Werte $c$ .................................. 34
Abbildung 15: Pruningmethoden aus [Fürn97] .......................................................... 35
Abbildung 16: Postpruning Algorithmus [Fürn97] ..................................................... 37
Abbildung 17: FOIL-MDL Anomalie ........................................................................... 40
Abbildung 18: Likelihood Ratio Statistic Isometrien ............................................... 42
Abbildung 19: Pseudocode IREP ............................................................................ 43
Abbildung 20: Ein generischer SeCo-Pruning-Algorithmus ....................................... 47
Abbildung 21: Pakete des SeCo-Frameworks ............................................................. 48
Abbildung 22: Klasse PruningTemplate .................................................................... 49
Abbildung 23: Klasse RuleSet- und RuleOperator .................................................... 50
Abbildung 24: Interfaces für Stopkriterien ............................................................... 50
Abbildung 25: Instanzierte Methoden für das Prepruning ........................................ 52
Abbildung 26: Instanzierte Methoden für IREP ........................................................ 53
Abbildung 27: Instanzierte Methoden für REP .......................................................... 54
Abbildung 28: Instanzierte Methoden für IREPOpt ................................................... 55
Abbildung 29: Durchschnittliche Genauigkeit der Covering- und JRip-Konfigurationen auf den Datenpaketen ...................................................................... 63
Abbildung 30: durchschnittliche Genauigkeiten des Preprunings auf dem 0% Paket ................................................................. 65
Abbildung 31: durchschnittliche Genauigkeiten des Preprunings auf dem 5% Paket ................................................................. 67
Abbildung 32: durchschnittliche Genauigkeiten des Preprunings auf dem 10% Paket ................................................................. 69
Abbildung 33: durchschnittliche Genauigkeiten im Vergleich zum Covering Algorithmus für die IREP

Konfigurationen mit find-best-simplification Operator .................................... Fehler! Textmarke nicht definiert.
Abbildung 34: durchschnittliche Genauigkeiten im Vergleich zum Covering Algorithmus für die IREP

Konfigurationen mit delete-last-condition Operator ......................................... Fehler! Textmarke nicht definiert.
# Tabellenverzeichnis

Tabelle 1: Beispiel einer Trainingsmenge. Teile aus [Mite97]. ................................................................. 11
Tabelle 2: Beispiel - Genauigkeiten von Ripper und Covering ............................................................... 22
Tabelle 3: Merkmale einer Regel ........................................................................................................ 23
Tabelle 4: Merkmale einer Trainingsmenge ......................................................................................... 23
Tabelle 5: Abgeleitete Merkmale einer Regel bzw. Regelmenge ......................................................... 23
Tabelle 6: Konfusionsmatrix ................................................................................................................. 23
Tabelle 7: Merkmale ROC- vs. PN-Raum. (übernommen aus [FüFl05]) .................................................. 24
Tabelle 8: $\chi^2$-2-Test Tabelle für einen Freiheitsgrad [Wiki08] ................................................................. 41
Tabelle 9: Beispiel LRS und Chi-Quadrat ............................................................................................. 41
Tabelle 10: SeCo-Framework Pakete .................................................................................................. 46
Tabelle 11: Variable Prozeduren für das Prunen. Teile aus [THIE05] ...................................................... 47
Tabelle 12: Teil 1 der verwendeten Datensätze ..................................................................................... 56
Tabelle 13: Teil 2 der verwendeten Datensätze ..................................................................................... 57
Tabelle 14: JRip Konfigurationen ......................................................................................................... 58
Tabelle 15: Covering Konfigurationen .................................................................................................. 59
Tabelle 16: Genauigkeiten Prepruning mit CutOff Kriterium. Mit und ohne Test auf fp<tp ..................... 59
Tabelle 17: Durchschnittliches Zeitverhalten, durchschnittliche Hypothesengröße und Genauigkeit von JRip und Covering .................................................................................................................. 62
Tabelle 41: Zusammenfassung der besten Konfigurationen ................................................................. Fehler! Textmarke nicht definiert.
Tabelle 42: Win/Loss der besten Konfigurationen auf dem 0%-Paket ...................................................... Fehler! Textmarke nicht definiert.
Tabelle 43: Win/Loss der besten Konfigurationen auf dem 5%-Paket .................................................... Fehler! Textmarke nicht definiert.
Tabelle 44: Win/Loss der besten Konfigurationen auf dem 10%-Paket ................................................ Fehler! Textmarke nicht definiert.
Kapitel 1: Einleitung

1.1 Motivation und Ziel


„Ist Pruning überhaupt noch notwendig, wenn die Heuristiken, die zum Lernen verwendet werden, schon sehr gute Ergebnisse erzielen?“

Ziel dieser Arbeit ist es die Pruningalgorithmen in einem einheitlichen Framework zu implementieren und zu untersuchen, inwiefern das Pruning in der Lage ist die Ergebnisse, die durch die Heuristiken erzielt werden, noch zu verbessern oder ob das Pruning keine Verbesserung mehr erzielen kann. Untersucht werden dazu das Prepruning und das inkrementelle Pre- und Postpruning in Verbindung mit den Heuristiken Laplace, Precision, Accuracy, Weighted Relative Accuracy, m-Estimate, Klösgen-Maß und Correlation. Für das m-Estimate und das Kloesgen-Maß werden insgesamt drei verschiedenen Parametereinstellungen betrachtet. Für die Pruningalgorithmen werden vier verschiedene Stopkriterien und zwei Pruningoperatoren genauer betrachtet. Um einen möglichst aussagekräftigen Vergleich zu erhalten werden die verschiedenen Algorithmenkonfigurationen auf insgesamt 57 nicht verrauschten Datensätzen getestet. Um auch eine Aussage über die Güte der gelernten (und geprunten) Hypothesen für verrauschte Daten treffen zu können, werden die 57 Datensätze zusätzlich mit 5% und 10% Rauschen versehen, womit insgesamt 171 zur Verfügung stehen. Die Algorithmen werden dann anhand ihres Zeitverhaltens, der Hypothesengröße und der durchschnittlichen erzielten Genauigkeit

1 Die Worte Pruning und Prunen werden im weiteren Verlauf der Arbeit synonym verwendet.
vergleichen. Um festzustellen welcher Algorithmus nun der Beste ist, werden die Ergebnisse auch anhand einer Win-Tie-Loss Tabelle festgehalten.

1.2 Überblick über die Kapitel

Das zweite Kapitel widmet sich den Grundlagen der Separate-and-Conquer Algorithmen. Zu Beginn des Kapitels wird die grundlegende Problemstellung des Lernens beschrieben. Im weiteren Verlauf werden die Separate-and-Conquer Algorithmen und ihre Merkmale (Kapitel 2.2) beschrieben und anhand eines generischen Algorithmus beschrieben (Kapitel 2.3). Am Ende des Kapitels (Kapitel 2.4) werden Methoden zum Vergleich von Lernalgorithmen kurz vorgestellt.

Das nachfolgende Kapitel (Kapitel 3) bietet einen kurzen Überblick über Heuristiken. Dazu werden wichtige Merkmale von Regeln und Trainingsmengen definiert (Kapitel 3.1). Im weiteren Verlauf des Kapitels werden PN- bzw. ROC-Räume (Kapitel 3.2) vorgestellt, die für die Visualisierung der Heuristiken verwendet werden. Abschließend werden die Heuristiken, die in dieser Arbeit verwendet wurden charakterisiert und visualisiert (Kapitel 3.3).


Kapitel 5 beschreibt die Umsetzung der Pruningmechanismen im SeCo-Framework. Dazu wird ein generischer Separate-and-Conquer Pruningalgorithmus definiert und auf den generischen Algorithmus des Frameworks übertragen (Kapitel 5.1). Im weiteren Verlauf des Kapitels wird definiert, welche Objekte benötigt werden um das Pruning in das Framework zu integrieren und wie die Integration bewerkstelligt wurde (Kapitel 5.2). Am Ende des Kapitels wird die Integration von vier verschiedenen Algorithmen erläutert (Kapitel 5.4 bis Kapitel 5.7).

Das sechste Kapitel beschreibt ausführlich die verwendeten Testdatensätze und die verschiedenen Konfigurationen der Algorithmen, die in dieser Arbeit verglichen werden (Kapitel 6.1 und Kapitel 6.2). Der Rest des Kapitels dient dem Vergleich der Algorithmen untereinander. Am Ende des Kapitels werden die Ergebnisse zusammengefasst und es wird versucht die Frage zu beantworten:

„Ist Pruning überhaupt noch notwendig, wenn die Heuristiken, die zum Lernen verwendet werden, schon sehr gute Ergebnisse erzielen?“

Kapitel 7 fasst nochmals kurz die gesamten Ergebnisse der Arbeit zusammen und weist auf Probleme während dem Verlauf der Arbeit hin. Abschließend werden Ausblicke für neue interessante Ansätze dargestellt.
Kapitel 2: Separate-and-Conquer Regellernen

In diesem Kapitel wird zuerst das grundlegende Lernproblem erläutert und grundlegende Definitionen vorgenommen. Anschließend wird das Lernproblem anhand der Separate-And-Conquer Strategie genauer beleuchtet. Die Strategie wird weiter durch einen generischen Algorithmus beschrieben und eine Unterscheidung der SeCo – Algorithmen wird entlang von drei Dimensionen (Verwendete Hypothesensprache, Suchalgorithmus für Hypothesen und Vermeidung von Overfitting) gemacht, wobei das Vermeiden von Overfitting hier ausgespart wird und im Kapitel 3 ausführlich erklärt wird.

2.1 Induktives Lernen

Das induktive Lernen beschäftigt sich mit dem Problem wie man eine allgemeine Funktion \( h'(\bar{e}) \) aus einer Menge \( E \) von spezifischen Trainingsdaten findet. Die Trainingsdaten enthalten die Eingabewerte \( \bar{e} \) verknüpft mit den korrekten Ausgabewerte \( h(\bar{e}) \) der Zielfunktion. Im Allgemeinen ist die Zielfunktion eine boolesche Funktion, die einer bestimmten Repräsentationssprache unterliegt, und wird als Zielhypothese bzw. Zielkonzept bezeichnet. Anhand der Ausgabewerte \( h(\bar{e}) \) lassen sich die Beispiele in zwei Mengen, auch Klassen genannt, einteilen. Die Klassen geben an, welches Beispiel zum Zielkonzept gehört und welche nicht. Damit können die Trainingsdaten in zwei Mengen aufgeteilt werden, nämlich die Menge der positiven Beispiele \( E^\Theta \), für die \( h(\bar{e}) = 1 \) ist und zum Zielkonzept gehören, und die Menge der negativen Beispiele \( E^\ominus \), für die \( h(\bar{e}) = 0 \) ist und nicht zum Zielkonzept gehören. Im Folgenden wird angenommen, dass sich das Lernen auf 2-Klassen-Probleme beschränkt. In Kapitel 2.3 werden Methoden vorgestellt, wie das Lernen von Mehr-Klassen-Problemen funktioniert.

Wie zuvor schon durch den Vektorpfeil über einem Beispiel \( \bar{e} \) angedeutet, ist ein Beispiel aus der Trainingsmenge ein Vektor von Attributen, wobei jedem Attribut ein Wert aus dessen Wertebereich zugeordnet ist.

**Definition 2.1.1 (Attribut):** Ein Attribut \( A \) besitzt einen Namen, einen Wertebereich und ist einer Attributklasse zugeordnet.


**Definition 2.1.2 (Beispiel, Instanz):** Ein Beispiel \( \bar{e} \) ist ein Vektor von Attributen \( (A_1, ..., A_n) \), \( n \in \mathbb{N} \). Den Attributen ist dabei ein bestimmter Wert aus ihrem Wertebereich zugewiesen, wobei das letzte Attribut das Klassenattribut ist:

\[
\bar{e} = (A_1 = Wert_1, ..., A_{n-1} = Wert_{n-1}, A_n = h) \quad (2.1)
\]

Die Gleichheitszeichen bedeuten im Beispielvektor eine Wertzuweisung und sind nicht als Vergleich zu betrachten. Anhand der Definition der Beispiele lässt sich nun definieren was die Beispiel- bzw. Trainingsmenge ist.
**Definition 2.1.3 (Trainingsmenge, Beispielmenge):** Die Trainingsmenge \( E \) ist eine Menge von Beispielen \( \bar{e}_k^1, k \in \mathbb{N} \).

\[ E = \{ \bar{e}_1, \bar{e}_2, \bar{e}_3, \ldots, \bar{e}_k \} \quad (2.2) \]

Eine Trainingsmenge könnte somit folgendermaßen aussehen:

<table>
<thead>
<tr>
<th>Beispiel</th>
<th>Himmel</th>
<th>Temperatur</th>
<th>Luftfeuchtigkeit</th>
<th>Wasser</th>
<th>Wind</th>
<th>Schwimmen gehen?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sonnig</td>
<td>20</td>
<td>normal</td>
<td>warm</td>
<td>stark</td>
<td>ja</td>
</tr>
<tr>
<td>2</td>
<td>sonnig</td>
<td>22</td>
<td>hoch</td>
<td>warm</td>
<td>stark</td>
<td>ja</td>
</tr>
<tr>
<td>3</td>
<td>regnerisch</td>
<td>14</td>
<td>hoch</td>
<td>warm</td>
<td>stark</td>
<td>nein</td>
</tr>
<tr>
<td>4</td>
<td>sonnig</td>
<td>19</td>
<td>hoch</td>
<td>kalt</td>
<td>stark</td>
<td>ja</td>
</tr>
</tbody>
</table>

Tabelle 1: Beispiel einer Trainingsmenge. Teile aus [Mitc97].

Im Allgemeinen wird bei der Beschreibung von Beispielen in den Vektorkomponenten nur der Wert des jeweiligen Attributes angegeben. Ein Beispiel aus der Trainingsmenge von Tabelle 2.1 sieht damit so aus:

\[(sonnig, 20, normal, warm, stark, ja)\] (2.3)

In der obigen Trainingsmenge sind die Attribute der Beispiele „Himmel, Temperatur, Luftfeuchtigkeit, Wind und Wasser“, die Klasse ist „Schwimmen gehen?“. Das Attribut Temperatur ist ein numerisches Attribut, wohingegen die anderen Attribute und das Klassenattribut ausschließlich nominale Attribute sind. Eine Besonderheit des Klassenattributes hier ist, dass es ein binäres Attribut ist. Es besteht nur aus zwei Werten („ja“, „nein“).

Das Lernen einer Funktion \( h' (\bar{e}) \), auch Hypothese bzw. Konzept genannt, kann als Suche durch den Raum aller möglichen Hypothesen betrachtet werden. Ziel eines induktiven Lernalgorithmus ist es dann eine Hypothese zu finden, die möglichst genau die Trainingsdaten beschreibt. D.h., dass die gefundene Hypothese vollständig (Definition 2.1.4) und konsistent (Definition 2.1.5) auf den Trainingsdaten ist.

**Definition 2.1.4 (Konsistenz):** Eine Hypothese ist konsistent auf den Trainingsdaten, wenn sie keine negativen Beispiele abdeckt.

**Definition 2.1.5 (Vollständigkeit):** Eine Hypothese ist vollständig, wenn sie alle positiven Beispiele abdeckt.

Die so gefundene Hypothese wird zur Klassifikation von Beispielen benutzt, die nicht in der Trainingsmenge enthalten waren bzw. dessen Klassenwert nicht bekannt ist. Als Grundlage für das Klassifizieren von nicht bekannten Beispielen dient folgende Annahme:

**Annahme 2.1.6 (Induktives Lernen Hypothese):** Eine Hypothese, die die Zielhypothese auf einer ausreichend großen Menge von Trainingsdaten hinreichend genau approximiert, approximiert die Zielhypothese auch auf vorher nicht bekannten Beispielen hinreichend genau [Mitc97].

Um festzustellen wie gut die gefundene Hypothese die Zielhypothese annähert, klassifiziert man mit der gefundenen Hypothese Beispiele aus einer separaten Testmenge. Die Testmenge enthält Beispiele, für die der korrekte Wert der Zielhypothese bekannt ist, aber dem Lernalgorithmus beim Lernen vorenthalten wurde. Anhand der richtig bzw. falsch klassifizierten Beispiele in der Testmenge, kann man die Genauigkeit (Definition 2.1.7) bzw. den Fehler (Definition 2.1.8) einer Hypothese errechnen:
Definition 2.1.7 (Genauigkeit): \[ \text{Genauigkeit} = \frac{\text{Anzahl korrekt klassifizierter Beispiele}}{\text{Anzahl aller Beispiele}} \]

Definition 2.1.8 (Fehler): \[ \text{Fehler} = \frac{\text{Anzahl falsch klassifizierter Beispiele}}{\text{Anzahl aller Beispiele}} = 1 - \text{Genauigkeit} \]

Im Regelfall steht eine solche Testmenge nicht zur Verfügung, sodass man nur die Genauigkeit bzw. den Fehler auf den Trainingsdaten hat. Um dennoch eine verlässliche Aussage über die gelernte Hypothese treffen zu können, bietet es sich an, die Trainingsdaten vor dem Lernen aufzuteilen, damit eine separate Testmenge vorhanden ist. Eine andere Möglichkeit ist die Kreuzvalidierung (Crossvalidation) an.

Definition 2.1.9 (Kreuzvalidierung, Crossvalidation): Bei einer Crossvalidation wird die Trainingsmenge in n gleichgroße Teile aufgeteilt. Ein Teil der Aufteilung wird als Testmenge benutzt, die anderen n-1 Teile werden benutzt um eine Hypothese zu finden. Insgesamt werden n Hypothesen gelernt, wobei jeder Teil einmal als Testmenge benutzt wird, und die Ergebnisse des Klassifizierens werden gemittelt.


Ist die Genauigkeit auf der Testmenge niedrig bzw. der Fehler hoch, so deutet das daraufhin, dass es während dem Lernen zu einer Überanpassung der Hypothese an die Trainingsdaten gekommen ist oder die Trainingsmenge verrauscht ist. Die so gefundene Hypothese ist meist wenig aussagekräftig. Man sagt, dass die Hypothese nicht gut verallgemeinert. Dies kann mehrere Gründe haben. Zum einen die oben genannten oder zum anderen die Aufteilung der Beispielmenge in Trainings- und Testmenge oder die Aufteilung der Beispielmenge bei einer Crossvalidation. Es kann passieren, dass sich dann wichtige Beispiele, die benötigt werden um die Zielhypothese zu beschreiben, in der Testmenge befinden.

Definition 2.1.10 (Overfitting, Überangepasstheit): Für eine gegebene Menge an Trainingsdaten und genügend Freiheitsgraden, lässt sich immer eine Hypothese finden, die die Trainingsdaten genau beschreibt. Die Trainingsdaten können dabei korrekt oder fehlerhaft sein.

Definition 2.1.11 (Noise, Rauschen): Fehler in der Trainingsmenge, wie falsche oder fehlende Wert von Beispielen.

Um ein Overfitting zu vermeiden oder die Auswirkung von verrauschten Daten zu vermindern, kann man versuchen die gefundene Hypothese noch zu verbessern oder die Konsistenz- und Vollständigkeitsnebenbedingungen zu lockern, sodass auch einige negative Beispiele abgedeckt werden dürfen. Dieser Vorgang wird im Allgemeinen als Pruning bezeichnet. Eine so gelernte Hypothese ist im Allgemeinen kleiner als eine überangepasste Hypothese. Die Hypothese in Abbildung 2 besteht aus 7 Regeln, wohingegen die Hypothese aus Abbildung 3 nur 3 Regeln umfasst.

Die nachfolgenden Abschnitte behandeln die Merkmale von Separate-and-Conquer Algorithmen genauer. Der Abschnitt 2.2.1 beleuchtet eine spezielle Hypothesensprache, die in dieser Arbeit benutzt wird, genauer. Der Schwerpunkt des nächsten Abschnitts (2.2.2) liegt auf den verschiedenen Suchverfahren, die benutzt werden um eine Hypothese zu lernen. Das Suchverfahren gliedert sich in drei Bereiche, den Suchalgorithmus, die Suchheuristik und die Suchstrategie. Die Suchheuristiken und ihre Charakteristika werden noch genauer in Kapitel 3 beleuchtet. Das Vermeiden von Overfitting wird kurz in Abschnitt 2.2.3 und genauer in Kapitel 4 beschrieben. Im Anschluss an diese Abschnitte, im Abschnitt 2.4, wird ein generischer Separate-and-Conquer Algorithmus vorgestellt und erklärt wie
die verschiedenen Merkmale implementiert werden können. Der letzte Abschnitt (2.5) befasst sich damit, wie man verschiedene Lernalgorithmen vergleichen kann.

2.2 Merkmale von Separate-and-Conquer Algorithmen

2.2.1 Hypothesensprache
Die erste Dimension an der Separate-and-Conquer Algorithmen charakterisiert werden ist die Hypothesensprache, die dem Algorithmus zugrunde liegt, und das zu lernende Konzept beschreibt. Ein Separate-and-Conquer Algorithmus lernt Regeln zur Beschreibung der Zielhypothese. Die wichtigsten Hypothesensprachen, die man bei Separate-and-Conquer Algorithmen unterscheiden kann, sind dabei Regelmengen in disjunktiver Normalform (DNF) (Definition 2.2.5) oder konjunktiver Normalform (KNF) (Definition 2.2.6), Entscheidungslisten, Logikprogramme, Regressionsregeln oder Funktionale Relationen [Fürn99], je nachdem welche Zielhypothese gelernt werden soll. Für kontinuierliche bzw. numerische Zielhypothesen eignen sich vor allem Regressionsregeln, für nominale Zielhypothesen eignen sich Regelmengen bzw. Entscheidungslisten.


Die Hypothesensprache, die in dieser Arbeit verwendet wird, beschränkt sich auf Regelmengen, mit besonderem Augenmerk auf Entscheidungslisten, da der Algorithmus der benutzt wird auf Entscheidungslisten basiert. Dazu werden hier grundlegende Begriffe, wie Regel, Regelmenge, etc., festgelegt um dann zu definieren was eine Entscheidungsliste ist.

Entscheidungslisten und Regelmengen bestehen aus Regeln, die im Folgenden definiert werden.

**Definition 2.2.1 (Regel):** Eine Regel \( r \) besteht aus einem Kopf und einem Körper (Gleichung 2.3). Der Regelkörper besteht aus Attributtests (siehe Definition 2.2.5), die entweder konjunktiv oder disjunktiv miteinander verknüpft sind. Der Regelkopf besteht aus der Vorhersage eines Klassenwerts.

\[ r: \text{Regelkörper} = \neg \text{Regelkopf} \ (2.3) \]

**Definition 2.2.2 (Attributtest):** Ein Attributtest ist ein Vergleich eines Attributes mit einem Wert aus dem Wertebereich des Attributes, der fordert, dass das Attribut mit diesem Wert übereinstimmt oder dieser Wert den Wertebereich des Attributs beschränkt.

Für Attributtest werden verschiedene Vergleichsoperatoren, auch Komparatoren genannt, verwendet. Die wichtigsten Komparatoren sind für nominale Attribute der Gleichheitsoperator \( = \) bzw. der Ungleichheitsoperator \( \neq \). Für numerische Attribute gelten die Komparatoren \( <, \leq, > \text{ und } \geq \), die den Wertebereich eines Attributes einschränken. Eine Regel könnte damit wie in Gleichung 2.4 aussehen.

\[ \text{Wasser} = \text{warm} \land \text{Temperatur} \geq 14 \land \text{Wind} = \text{stark} : \neg \text{Sport macht Spaß}? = \text{ja} \ (2.4) \]

Der Klassencode des Regelkopfs wird für ein Beispiel vorhergesagt (Definition 2.2.3), wenn die Regel das Beispiel abdeckt (Definition 2.2.4).

**Definition 2.2.3 (Vorhersagen):** Wird ein Beispiel von einer Regel abgedeckt, so wird der Wert im Regelkopf für das Beispiel vorhergesagt.

**Definition 2.2.4 (Abdecken):** Eine Regel deckt ein Beispiel ab, wenn der Körper der Regel wahr ist. Also wenn jeder Attribut-Wert-Vergleich wahr ist bei konjunktiven Regeln.

Mit diesen einfachen Definitionen lassen sich nun Regelmengen (Definition 2.2.7) definieren. Eine Regelmengen ist entweder in DNF (Definition 2.2.5) oder in KNF (Definition 2.2.6). Beide Repräsentationen sind äquivalent, da sich die DNF durch zweifache Negation in die KNF und die KNF durch zweifache Negation in DNF überführen lässt.

**Definition 2.2.5 (disjunktive Normalform, DNF):** Die disjunktive Normalform ist eine Disjunktion von konjunktiv verknüpften Literalen \( L_{i,j} \). Die Literale \( L_{i,j} \) bestehen dabei aus Attributtests:

\[ \bigvee_i \bigwedge_j L_{i,j} = (L_{1,1} \land L_{1,2} \land ... \land L_{1,j}) \lor ... \lor (L_{i,1} \land L_{i,2} \land ... \land L_{i,j}) \ (2.5) \]

**Definition 2.2.6 (konjunktive Normalform, KNF):** Die konjunktive Normalform ist eine Konjunktion von disjunktiv verknüpften Literalen \( L_{i,j} \). Die Literale \( L_{i,j} \) bestehen dabei aus Attributtests:

\[ \bigwedge_i \bigvee_j L_{i,j} = (L_{1,1} \lor L_{1,2} \lor ... \lor L_{1,j}) \land ... \land (L_{i,1} \lor L_{i,2} \lor ... \lor L_{i,j}) \ (2.6) \]
Definition 2.2.7 (Regelmenge): Eine Regelmenge ist eine Menge von Regeln, die entweder in KNF oder in DNF ist. Eine Regelmenge in KNF (Gleichung 2.7) besteht aus konjunktiv verknüpften Regeln, deren Attributtests disjunktiv verknüpft sind. Eine Regelmenge in DNF (Gleichung 2.6) besteht aus disjunktiv verknüpften Regeln, deren Attributtests konjunktiv verknüpft sind.

\[ R = \{ r_1 \lor r_2 \lor \ldots \lor r_n \} \] (2.6)

\[ R = \{ r_1 \land r_2 \land \ldots \land r_n \} \] (2.7)

Die Entscheidungsliste ist auch eine Regelmenge. Allerdings ist diese Regelmenge sortiert und die Reihenfolge, in der die Regeln in der Regelmenge enthalten sind, ist wichtig. Bei der Klassifikation von Beispielen wird der Reihe nach für jede Regel geprüft, ob diese das Beispiel abdeckt. Deckt eine Regel ein Beispiel ab, so wird der Klassenwert der Regel für das Beispiel vorhergesagt und die übrigen Regeln werden ignoriert. Desweiteren wird eine spezielle Regel, die sogenannte Defaultregel (Definition 2.2.8) die die Defaultklasse (Definition 2.2.9) für ein Beispiel vorhersagt, zu der Entscheidungsliste hinzugefügt.

Definition 2.2.8 (Defaultregel): Eine Defaultregel ist eine Regel deren Regelkörper keine Attributtests enthält, sondern nur den Wert true. Die Defaultregel deckt jedes Beispiel ab und sagt für ein Beispiel eine Defaultklasse (Definition 2.2.6) vorher.

Definition 2.2.9 (Defaultklasse): Die Defaultklasse ist die Klasse, deren Klassenwert am häufigsten in der Trainingsmenge vorkommt. Die Defaultklasse wird auch als Majorityklasse bezeichnet.

Mit den vorangegangenen Definitionen lässt sich eine Entscheidungsliste folgendermaßen definieren:


\[ R = \begin{cases} 
  r_1 \\
  r_2 \\
  \vdots \\
  r_n \\
  \text{true} : \neg \text{Defaultklasse} 
\end{cases} \] (2.8)

In den nachfolgenden Kapiteln beschränken sich die Beschreibungen und Definitionen auf konjunktive Regeln und Entscheidungslisten, da die Algorithmen, die in dieser Arbeit verglichen werden auf dieser Hypothesensprache basieren.

2.2.2 Suchverfahren

Der Suchalgorithmus steuert, wie der Name schon impliziert, die Suche. Eine Option ist es, einfach alle Regeln, die bei der Suche nach einer Hypothese erzeugt werden, zu generieren und nur die Regeln zu nehmen, die nur positive Beispiele abdecken. Eine andere Option wäre den Hypothesenraum erschöpfend nach einer Hypothese zu durchsuchen. Beide Optionen sind aber sehr ineffizient und werden meist nur zu theoretischen Zwecken implementiert. [Fürn99]

Die Algorithmen, die hauptsächlich bei Separate-and-Conquer Algorithmen verwendet werden, sind Hill-Climbing, Beam Search, Best-First Search oder Stochastische Suche. Das Hill-Climbing durchsucht den Hypothesenraum, indem es eine Regel sukzessive verfeinert bis ein Optimum erreicht ist. Die Regel wird dazu entweder generalisiert (Definition 2.2.11) oder spezialisiert (Definition 2.2.12). Dabei erzeugt die Suche alle möglichen Regeln, die von der momentanen Regel aus erzeugbar sind. Aus den erzeugten Regeln wird die beste Regel ausgewählt und die Suche startet erneut mit der besten Regel als Startpunkt.

**Definition 2.2.11 (Generalisierung):** Eine Regel wird generalisiert, indem ein oder mehrere Attributtests aus dem Regelkörper entfernt werden. Die Regel deckt somit mehr Beispiele ab.

**Definition 2.2.12 (Spezialisierung):** Eine Regel wird spezialisiert, indem ein oder mehrere Attributtests dem Regelkörper hinzugefügt werden. Die Regel deckt somit weniger Beispiele ab.

Die Hill-Climbing Suche versucht zwar ein globales Optimum zu finden, kann aber auch leicht in einem lokalen Optimum stecken bleiben. Dies geschieht, da die Suche nur einen Schritt voraussieht, d.h. die beste erzeugte Regel weiter verfeinert und die anderen Regeln vergisst. Ist das globale Optimum aber eine Verfeinerung einer vergessenen Regel, so läuft die Suche am globalen Optimum vorbei. Um diese „Kurzsichtigkeit“ der Suche zu umgehen wird die Suche so verändert, dass in einem Schritt nicht nur eine sondern \( n \) Regeln simultan verfeinert werden.

Im Gegensatz zum Hill-Climbing speichert die Beam-Suche nicht nur eine verfeinerte Regel, sondern die \( b \) besten Regeln in dem sogenannten Strahl (Beam). Mit der Beam-Suche will man die Auswirkung der Kurzsichtigkeit einer Hill-Climbing Suche vermindern. Dadurch, dass man einen größeren Teil des Hypothesenraums durchsucht, ist die Wahrscheinlichkeit ein globales Optimum zu finden höher bzw. die Wahrscheinlichkeit in einem lokalen Optimum festzuhängen geringer. Setzt man allerdings die Beamgröße auf \( b = 1 \), so erhält man wieder eine Hill-Climbing Suche. Wählt man dagegen einen zu großen Strahl, dann verschlechtern sich die Ergebnisse der Beam-Suche je nach verwendeter Suchheuristik [JaFü08].

Wird die Beamgröße bei der Beam-Suche auf \( b = \infty \) festgelegt, kommt man zur Best-First-Suche. Dies führt dazu, dass der Suchraum erschöpfend durchsucht wird. Dieser Algorithmus vermeidet das Problem einer zu kurzzeitigen Suche und findet eine optimale Hypothese. Eine erschöpfende Suche ist aber sehr ineffizient, da der gesamte Hypothesenraum durchsucht wird. Die Suche kann mit dem A*-Algorithmus weiter eingeschränkt werden, indem Teile des Suchraums nicht betrachtet werden ohne dabei die Optimalität der Hypothese zu verlieren [Fürn99, HaNR68].


Die letzte Möglichkeit die Suche zu steuern ist die Suchheuristik, die auch den meisten Einfluss auf die Suche hat. Die Suchheuristik schätzt die Güte einer Regel, die durch verschiedene Eigenschaften der Regel festgelegt wird, und führt die Suche in die richtige Richtung im Suchraum. Die meisten Heuristiken schätzen die Güte einer Regel anhand der positiven und negativen Beispiele, die eine Regel abdeckt. Diese Abschätzungen können allerdings zu optimistisch ausfallen, da sie auf den Trainingsdaten gemacht werden und sich die Trainingsmenge von der Testmenge unterscheiden kann, d.h. die zugrunde liegende Hypothese, die zur Erstellung beider Mengen verwendet wurde kann unterschiedlich sein. Eine andere Variante um die Güte einer Regel zu schätzen wurde in [JaFu07] verfolgt. Hier wurde ein Neuronales Netz trainiert um die Genauigkeit der auf den Trainingsdaten gelernten Regeln auf der Testmenge vorherzusagen. Dieses Netz wurde dann als Suchheuristik verwendet. In Kapitel 3 werden Suchheuristiken genauer erläutert.

2.2.3 Methoden zu Vermeidung von Overfitting

Ein Separate-and-Conquer Algorithmus ist bis jetzt in der Lage konsistente und vollständige Theorien zu lernen. Was passiert aber wenn die Daten verrauscht sind oder die Trainingsdaten nur einen kleinen Teil der Zielhypothese beschreiben? In diesem Fall beschreibt die gelernte Hypothese die Trainingsdaten genau, verallgemeinert aber schlecht auf vorher nicht gesehenen Beispielen. Um diese Überanpassung zu vermeiden versucht man die gelernte Hypothese zu prunen. D.h. dass der Algorithmus versucht nicht nur konsistente und vollständige Regeln zu lernen, sondern auch Regeln, die einige negative Beispiele abdecken. Die so gelernte Hypothese deckt fast alle positiven und einige negative Beispiele ab. Es gibt mehrere Möglichkeiten um eine Hypothese zu prunen. Zum einen kann eine geeignete Suchheuristik während des Lernens dazu beitragen. In der Regel bevorzugen Heuristiken kurze, allgemeine Regel gegenüber langen, speziellen Regeln, auch wenn die Genauigkeit
der kurzen Regeln schlechter als die Genauigkeit der speziellen Regeln ist, in der Hoffnung, dass die Genauigkeit der allgemeinen Regeln auf nicht gesehenen Daten höher ist [Fürn99]. Zum anderen gibt es auch verschiedene Methoden, wie z.B. Pre- und Post-Pruning, die helfen eine überangepasste Theorie wieder zu verallgemeinern. Die verschiedenen Arten des Prunings werden im Kapitel 4 ausführlich beschrieben.

**procedure** SeparateAndConquer(Examples)

Theory = ∅
while POSITIVE(Examples) ≠ ∅
    Rule = FindBestRule(Examples)
    Covered = COVER(Rule)
    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
    for Candidate ∈ Candidates
        Refinements = REFINERULE(Candidate, Examples)
        for Refinement ∈ Refinements
            Evaluation = EVALUATERULE(Refinement, Examples)
            unless STOPPINGCRITERION(Refinement, Evaluation, Examples)
                NewRule = <Evaluation, Refinement>
                Rules = INSERTSORT(NewRule, Rules)
                if NewRule > BestRule
                    BestRule = NewRule
                Rules = FILTERRULES(Rules, Examples)
        return(BestRule)

Abbildung 4: Ein generischer Separate-and-Conquer Algorithmus [Fürn99]

**2.3 Ein generischer Separate-and-Conquer Algorithmus**


Bis jetzt wurde angenommen, dass sich das Lernen ausschließlich auf 2-Klassen-Probleme beschränkt. Aber viele Lernprobleme lassen sich nicht mit zwei Klassen beschreiben, sondern werden durch drei, vier oder mehr, z.B. n Klassen beschrieben. Damit die Algorithmen mit Mehr-Klassen-Problemen umgehen können, ist es notwendig die n Klassen in eine Folge von 2-Klassen Problemen zu transformieren (Class binarization). Die erste Möglichkeit dies zu bewerkstelligen ist es Regeln ohne besonderes Augenmerk auf die Klassen zu lernen. Die Klasse die eine Regel für ein Beispiel

2.4 Vergleich von Separate-and-Conquer Algorithmen


Die erste Möglichkeit ist die Genauigkeit des Algorithmus festzustellen (Definition 2.1.7). Die Genauigkeit kann entweder auf den Trainingsdaten oder auf einer separaten Testmenge festgestellt werden. Da aber im Regelfall die Genauigkeit auf der Trainingsmenge sehr hoch ist und eine extra Testmenge meist nicht zu Verfügung steht, bietet sich hier eine Kreuzvalidierung (Definition 2.1.9) an.

Eine weitere Möglichkeit die Güte eines Algorithmus zu bestimmen ist die Größe der gelernten Hypothese. Eine kleine Hypothese, die alle positiven Beispiele abdeckt, ist natürlich einer großen Hypothese vorzuziehen. Die kleine Hypothese wird weniger an Overfitting leiden und ist leichter verständlich als eine große Hypothese. Die zugrundeliegenden Parameter, die hier als Gütemaß dienen, sind die Anzahl der Regeln und die Anzahl der Attributtest der Hypothese. Aus diesen Parametern lässt sich die durchschnittliche Regellänge errechnen, die ein weiteres Maß für die Güte eines Algorithmus ist.


2.4.1 Win-Tie-Loss Tabellen

Die Win-Tie-Loss Tabelle ist ein Mittel um mehrere Algorithmen miteinander zu vergleichen. Um die Tabellen zu erstellen lässt man mehrere Algorithmen auf einem oder mehreren Trainingsmengen lernen und stellt danach die Genauigkeit der Algorithmen fest. Im Anschluss daran zählt man wie oft ein Algorithmus besser (Win) bzw. schlechter (Loss) als ein anderer Algorithmus auf den Trainingsmengen ist. D.h. wie oft erreicht Algorithmus A eine höhere Genauigkeit als Algorithmus B auf der Beispielmenge $i, i \in \mathbb{N}$.

Anhand der Win-Tie-Loss Tabellen kann man mittels eines Vorzeichen-Tests (Sign-Test) ermitteln mit welcher Irrtumswahrscheinlichkeit ein Algorithmus A schlechter bzw. mit welcher Sicherheit ein Algorithmus A besser ist als ein anderer Algorithmus B. Der Vorzeichen-Test beantwortet die Frage, wie oft ein Algorithmus besser als ein Algorithmus B sein muss, damit mit einer bestimmten Sicherheit gesagt werden kann, dass A wirklich besser ist. Als Annahme gilt, dass beide Algorithmen, A und B, gleich sind. D.h. die Wahrscheinlichkeit, dass ein Algorithmus auf einem Datensatz gewinnt ist 50%. Die Irrtumswahrscheinlichkeit, dass ein Algorithmus mindestens (N-k)-mal bzw. höchstens k-mal gewinnt, lässt sich wie in Gleichung 2.9 berechnen. N ist die Anzahl der Experimente, wobei ein
Unentschieden nicht gezählt wird, und k gibt an wie oft ein Algorithmus A gegen einen Algorithmus B gewonnen hat.

\[
\text{Signtest: } P(i \leq k \lor i \geq N - k) = \frac{1}{2^{N-1}} \sum_{i=1}^{k} \binom{N}{i} \tag{2.9}
\]

Als Beispiel für den Vorzeichen-Test soll die Tabelle 2 dienen. Hier erkennt man, dass der Algorithmus Ripper im Vergleich zum Covering-Algorithmus auf allen Datensätzen, außer monk1, besser ist. Der Ripper-Algorithmus erzielt somit 8 Wins, 1 Loss und 0 Ties. Setzt man die Werte nun in Gleichung 2.9 ein so beträgt die Irrtumswahrscheinlichkeit \( p = 0.0352 \). Daraus kann man nun schließen, dass der Algorithmus RIPPER mit 95%-iger Sicherheit besser ist als der Covering-Algorithmus.

<table>
<thead>
<tr>
<th>Datensatz</th>
<th>Genauigkeit Ripper</th>
<th>Genauigkeit Covering</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance-scale</td>
<td>80,80 %</td>
<td>73,12 %</td>
</tr>
<tr>
<td>breast-cancer</td>
<td>70,89 %</td>
<td>67,48 %</td>
</tr>
<tr>
<td>colig</td>
<td>84,24 %</td>
<td>71,20 %</td>
</tr>
<tr>
<td>horse-colic</td>
<td>86,96 %</td>
<td>71,20 %</td>
</tr>
<tr>
<td>monk1</td>
<td>83,06 %</td>
<td>95,16 %</td>
</tr>
<tr>
<td>segment</td>
<td>95,71 %</td>
<td>93,90 %</td>
</tr>
<tr>
<td>vowel</td>
<td>70,81 %</td>
<td>49,49 %</td>
</tr>
<tr>
<td>zoo</td>
<td>86,14 %</td>
<td>85,15 %</td>
</tr>
</tbody>
</table>

Tabelle 2: Beispiel - Genauigkeiten von Ripper und Covering

\(^2\) Eine genaue Erklärung der Datensätze ist in Kapitel 6 zu finden.
\(^3\) Ripper wird genauer in Kapitel 4.4.2 besrieben.
\(^4\) Der Covering-Algorithmus wird in Kapitel 5 und 6 genauer betrachtet.
Kapitel 3 Suchheuristiken


3.1 Eigenschaften von Regeln, Regelmengen und Trainingsmengen

Eine Heuristik ordnet einer Regel $g$ einen bestimmten Wert $h(r)$ zu, der auf den Merkmalen der Regel bzw. den Merkmalen der Trainingsmenge, auf der die Regel gelernt wurde, basiert. Die Merkmale die den Trainingsdaten und einer Regel zugeordnet werden sind:

<table>
<thead>
<tr>
<th>$p$</th>
<th>Die Anzahl der abgedeckten positiven Beispiele</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Die Anzahl der abgedeckten negativen Beispiele</td>
</tr>
</tbody>
</table>

Tabelle 3: Merkmale einer Regel

<table>
<thead>
<tr>
<th>$P$</th>
<th>Die Anzahl aller positiven Beispiele in der Trainingsmenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Die Anzahl aller negativen Beispiele in der Trainingsmenge</td>
</tr>
<tr>
<td>$T = (P + N)$</td>
<td>Die Anzahl aller Beispiele in der Trainingsmenge</td>
</tr>
</tbody>
</table>

Tabelle 4: Merkmale einer Trainingsmenge

Aus diesen vier grundlegenden Eigenschaften lassen sich weitere Merkmale (siehe Tabelle 5) für Regeln ableiten.

| $tn = (N - n)$ | TrueNegative. Ein negatives Beispiel, dass als negatives Beispiel von einer Regel vorhergesagt wurde. |

Tabelle 5: Abgeleitete Merkmale einer Regel bzw. Regelmenge

<table>
<thead>
<tr>
<th>positives Beispiel</th>
<th>als positiv von der Regel vorhergesagt</th>
<th>als negativ von der Regel vorhergesagt</th>
</tr>
</thead>
<tbody>
<tr>
<td>$tp = p$</td>
<td>P</td>
<td></td>
</tr>
<tr>
<td>$fp = n$</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>$(p+n)$</td>
<td>$(P+N)-p$</td>
<td>$T = (P+N)$</td>
</tr>
</tbody>
</table>

Tabelle 6: Konfusionsmatrix

Die gleichen Merkmale für Regeln lassen sich auch einer Regelmenge zuorden. Im weiteren Verlauf gilt die Notation, die in den Tabellen 3 bis 5 benutzt wurde. Anhand der Merkmale lässt sich eine sogenannte Konfusions- oder Kontingenzmatrix erstellen.
Mit diesen grundlegenden Merkmalen lässt sich nun definieren was eine Heuristik ist:

**Definition 3.1.1 (Heuristik):** Eine Heuristik schätzt die Qualität bzw. Güte einer Regel, indem sie der Regel \( r \) einen Wert \( c \) zuordnet. Damit ist die Heuristik eine zweidimensionale Funktion, die eine Regel auf einen reellen Wert abbildet. Berechnet wird der Heuristikwert aus den Merkmalen einer Regel und/oder aus den Merkmalen der Trainingsmenge. Die Merkmale der Trainingsmenge können als konstant angesehen werden.

\[
H(r) = H(p, n) : \mathbb{N} \times \mathbb{N} \to \mathbb{R} \quad (3.1)
\]

**3.2 PN – und ROC – Raum**

Obwohl der ROC-Raum (Definition 3.2.1) seine Ursprünge in der Singaltheorie hat, besitzt er nützliche Eigenschaften um Hypothesen, einzelne Regeln oder den Lernprozess einer Hypothese bzw. Regel zu analysieren, zu evaluieren und zu visualisieren. Weitere Anwendungsgebiete des ROC-Raumes sind z.B. die Klassifikation von Beispielen selbst oder das Finden von Entscheidungsbaum Splitkriterien (decision tree splitting criteria) [Flac03].

**Definition 3.2.1 (ROC-Raum):** Der ROC-Raum ist durch die \( tpr \) (truepositivesrate) und \( fpr \) (falsepositivesrate) einer Hypothese bzw. einer Regel definiert. Ein Punkt \((x, y)\) im ROC-Raum entspricht einer Hypothese bzw. Regel mit einer \( tpr \) von \( y \) und \( fpr \) von \( x \).

Im Gegensatz zum ROC-Raum wird der PN-Raum (Definition 3.2.2) nicht durch die \( tpr \) und die \( fpr \) definiert, sondern von den absoluten Zahlen der positiven und negativen Beispielen einer Trainingsmenge.

**Definition 3.2.2 (PN-Raum):** Der PN-Raum wird durch die Anzahl der positiven \((P)\) und negativen \((N)\) Beispiele definiert. Ein Punkt \((x, y)\) im PN-Raum entspricht einer Hypothese bzw. Regel die \( x \) negative Beispiele \((n)\) und \( y \) positive Beispiele \((p)\) abdeckt.

Beide Räume lassen sich ineinander überführen, indem man den PN-Raum normalisiert, sodass die Achsen nur Werte zwischen 0 und 1 haben [FüFl03]. Der ROC-Raum lässt sich in den PN-Raum überführen, indem die \( tpr \)-Achse mit \( P \) und die \( fpr \)-Achse mit \( N \) multipliziert werden. Die unterschiedlichen Merkmale beider Räume sind in folgender Tabelle zusammengefasst:

<table>
<thead>
<tr>
<th>Merkmal</th>
<th>ROC-Raum</th>
<th>PN-Raum</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-Achse</td>
<td>( fpr = \frac{p}{N} )</td>
<td>( N )</td>
</tr>
<tr>
<td>y-Achse</td>
<td>( tpr = \frac{P}{p} )</td>
<td>( P )</td>
</tr>
<tr>
<td>Leere Theorie</td>
<td>(0, 0)</td>
<td>(0,0)</td>
</tr>
<tr>
<td>Korrekte Theorie</td>
<td>(0, 1)</td>
<td>(0, P)</td>
</tr>
<tr>
<td>Universelle Theorie</td>
<td>(1, 1)</td>
<td>(N, P)</td>
</tr>
<tr>
<td>Auflösung</td>
<td>( \frac{1}{N} \frac{1}{P} )</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>Steigung der Diagona</td>
<td>1</td>
<td>( P )</td>
</tr>
<tr>
<td>Steigung der Geraden ( p=n )</td>
<td>( \frac{N}{P} )</td>
<td>1</td>
</tr>
</tbody>
</table>

Tabelle 7: Merkmale ROC- vs. PN-Raum. (übernommen aus [FüFl05])

\[^5\) In diesem Sinne spricht man von Klassifikatoren
Im nachfolgenden beschränken sich die Betrachtungen auf den PN-Raum. Ausführliche Informationen zu ROC-Räumen finden sich in [FüFl05] und [Flac04].


In Abbildung 6 sind drei Punkte eingezeichnet, die jeweils einer Regel einer Hypothese entsprechen. Die Punkte geben an wie viele positive und negative Beispiele die Hypothese, nach dem Hinzufügen der Regel $R_i$, $i = 1, 2, 3$, abdeckt. Anhand des PN-Raumes lässt sich auch der Lernvorgang einer Hypothese visualisieren. Nehmen wir an, dass die Regeln in der Reihenfolge R1, R2, R3 zur Hypothese hinzugefügt wurden, dann ist der Startpunkt im PN-Raum beim Lernen (0, 0), was der leeren Theorie entspricht. Nachdem die Regel R1 zur Hypothese hinzugefügt wurde, deckt diese nun 30 negative und 90 positive Beispiele ab. Durch Hinzufügen der Regel R2 erhöht sich die Abdeckung der Hypothese auf den Trainingsdaten, der neu hinzugefügte Punkt tendiert in Richtung der universellen Theorie. Das gleiche passiert nachdem die dritte Regel hinzugefügt wurde. Jedes weitere Hinzufügen einer Regel nach der Dritten, würde die Trainingsdaten vollständig abdecken. Der Lernalgorithmus wäre bei der universellen Theorie, beim Punkt (N, P), angelangt. Man kann sich das Hinzufügen von Regeln zu einer Hypothese als Pfad durch den PN-Raum vorstellen. Genauso kann man sich das Lernen einer Regel als Pfad im PN-Raum vorstellen und visualisieren. Der Pfad beim Lernen einer Regel beginnt im Punkt (N, P) und bewegt sich in Richtung des Punktes (0, P).

Abbildung 5: PN-Raum Beispiel
Nachdem eine Regel \( R_i = (n_i, p_i) \) zu einer Hypothese hinzugefügt wurde, werden die Beispiele, die von dieser Regel abgedeckt werden, von der Trainingsmenge entfernt (Separate-Schritt). Für das Lernen einer neuen Regel \( R_{i+1} = (n_{i+1}, p_{i+1}) \) ist der PN-Raum deswegen kleiner als der vorangegangene PN-Raum \( PN_i \). Der Punkt \( R_i \) ist dann der Ursprung eines neuen, kleineren PN-Raums \( PN_{i+1} \) und das Lernen der Regel \( R_{i+1} \) ist nun auf einen PN-Raum der Größe \( N - n_i \) bzw. \( P - p_i \) beschränkt. Visualisiert man diese PN-Räume in einem einzigen Diagramm, so erhält man mehrere verschachtelte PN-Räume [FüFl05] (siehe Abbildung 7).

Abbildung 6: Beispiel Isometrik im PN-Raum

Abbildung 7: Verschachtelter PN-Raum (Accuracy Isometrien)
In Abbildung 7 sind verschiedene PN-Räume in einem PN-Raum eingezeichnet. Die Isometriken, die hier als rote gestrichelte Linien dargestellt sind, sind die Isometriken der Heuristik \textit{Accuracy}.

Das Ziel beim Lernen ist den Punkt (0, P), die optimale Theorie, zu erreichen, was aber in den wenigsten Fällen möglich ist. Das Lernen, wie im Beispiel, tendiert meist in Richtung des Punktes (N, P). Damit das Lernen in Richtung der optimalen Theorie tendiert, werden Heuristiken verwendet. Diese lassen sich durch sogenannte Isometriken (Definition 3.2.3) im PN-Raum visualisieren. Anhand der Isometriken lässt sich leicht die Tendenz beim Lernen hin zur korrekten Theorie visualisieren.

\textbf{Definition 3.2.3 (Isometrik):} Eine Isometrik einer Heuristik ist eine Kurve im PN-Raum, die alle Punkte (n, p), die den gleichen Heuristikwert h besitzen, verbindet [FuFl03]. Es gilt:

\[ H(r) = H(n, p) = h \ (3.2) \]

Die Isometriken sind eigentlich dreidimensionale Funktionen und werden im zweidimensionalen durch ihre Iso-Linien dargestellt. Die Iso-Linien sind zum Beispiel wie in Abbildung 6 oder 8 die roten gestrichelten Linien.

In Abbildung 6 sind die Isometriken der Heuristik aus Gleichung 3.4 zu sehen, die als rote gestrichelte Linien dargestellt sind. Zusätzlich zu den in Abbildung 6 eingetragenen Punkten sind noch zwei weitere Regeln (R4, R5) eingetragen, die einen im Vergleich zu den Regeln R1, R2 und R3, höheren bzw. niedrigeren Heuristikwert besitzen, und für jede Regel ist der Heuristikwert angegeben. Im Normalfall würde ein Lernalgorithmus beim Lernen die Regel R4 allen anderen Regeln vorziehen, da diese Regel den Heuristikwert der Heuristik aus Gleichung 3.3 maximieren würde.

Man erkennt hier leicht, dass ein Separate-and-Conquer Algorithmus versuchen wird Regeln auszuwählen, die einen möglichst hohen Heuristikwert haben, bzw. Regeln, die nahe am Punkt (0, P) sind. Anders ausgedrückt versucht der Algorithmus, die Anzahl der abgedeckten positiven Beispiele zu maximieren und die Anzahl der negativen Beispiele zu minimieren.

\section{3.3 Verwendete Heuristiken}


\subsection{3.3.1 Lineare Heuristiken}

Die linearen Heuristiken werden anhand der Linearen Kosten Metrik (Gleichung 3.3) oder \( h_{\text{costs}} - \text{Modell} \), und dem Precision Modell oder \( h_{pr} - \text{Modell} \), in zwei Familien unterteilt. Die Heuristik Precision (Gleichung 3.4) ist namensgebend für diese Familie von Heuristiken. Der Beweis der Äquivalenz der Heuristiken dieser Modelle ist in [FuFl05] zu finden.

\[ h_{\text{costs}} = a \ast p - b \ast n - c \ast p - (1 - c) \ast n - p - d \ast n \ (3.3) \]

\[ h_{pr} = \frac{p}{p + n} \ (3.4) \]
Das Hauptmerkmal der Heuristiken des $h_{\text{costs}} - \text{ModeMs}$ sind Isometrien, die parallel zueinander sind. Die Isometrien des $h_{\text{pr}} - \text{ModeMs}$ dagegen rotieren um einen Punkt im PN-Raum. Die Isometrien aus Abbildung 6 gehören damit also dem $h_{\text{pr}} - \text{ModeM}$ an. Ein weiterer Unterschied beider Modelle ist, dass die Isometrien für das $h_{\text{costs}} - \text{ModeMs}$ in verschachtelten PN-Räume immer die gleichen sind, wohingegen die Isometrien des $h_{\text{pr}} - \text{ModeMs}$ sich für jeden verschachtelten PN-Raum unterscheiden. Dies liegt daran, dass die Isometrien des $h_{\text{pr}} - \text{ModeMs}$ symmetrisch um einen Ursprung rotieren. Ein weiteres besonderes Merkmal der $h_{\text{costs}}$ Isometrien ist, dass ein lokales Optimum in einem verschachtelten PN-Raum auch gleichzeitig ein globales Optimum im eigentlichen PN-Raum ist. Wohingegen ein lokales Optimum des $h_{\text{pr}} - \text{ModeM}$ nicht immer ein globales Optimum ist [FüFl05].

**Heuristiken des $h_{\text{costs}} - \text{ModeMs}$:** Die grundlegendsten Heuristiken, die Basisheuristiken, dieses Modells sind MaximizePositives (Gleichung 3.5) und MinimizeNegatives (Gleichung 3.6).

\[
\text{MaximizePositives: } h_p(n, p) = p \quad (3.5) \\
\text{MinimizeNegatives: } h_n(n, p) = -n \quad (3.6)
\]

Die Isometrien der beiden Heuristiken aus Gleichung 3.5 und 3.6 sind in Abbildung 8 zu sehen. Die Isometrien auf der linken Seite sind die Isometrien der Heuristik MaximizePositives, die auf der linken Seite für MinimizeNegatives. Die Heuristik MaximizePositives bewertet Regeln, die sehr viele positive Beispiele abdecken, unabhängig davon wie viele negative Beispiele abgedeckt werden sehr gut. Die Heuristik MinimizeNegatives bewertet Regeln, die wenige negative Beispiele abdecken, unabhängig von der Anzahl der abgedeckten positiven Beispiele am besten. Für beide Heuristiken lässt sich schnell eine optimale Theorie bzw. Regel finden. $h_p$ ist maximal für die universelle Theorie und die korrekte Theorie. $h_n$ ist maximal für die leere Theorie, die keine negativen Beispiele abdeckt, aber auch keine positiven Beispiele. Eigentlich ist $h_p$ maximal für Theorien, die alle positiven Beispiele abdecken und $h_n$ ist maximal für Theorien, die keine negativen Beispiele abdecken. Beide Heuristiken lassen sich durch eine geeignete Wahl der Parameter aus Gleichung 3.5 herleiten. Dazu setzt man $a=0$ und $b=1$ für MinimizeNegatives oder $a=1$ und $b=0$ für MaximizePositives.

![Abbildung 8: MaximizePositives und MinimizeNegatives Isometrien](image)

Da das Ziel beim Lernen der Punkt $(0, P)$ ist, ist es sinnvoll beide Heuristiken, $h_p$ und $h_n$, gleichzeitig zu maximieren. Addiert man beide Heuristiken, erhält man daraus die Heuristik $h_{\text{acc}}$ (Gleichung 3.7), die äquivalent zu Accuracy ist [FüFl05]. Accuracy schätzt das Verhältnis der abgedeckten positiven ($p$) und nicht abgedeckten negativen Beispiele ($N-n$) einer Regel bzw. Hypothese zu allen Beispielen ($P+N$). Da $P$ und $N$ normalerweise konstant sind, kann man die Heuristik wie in (Gleichung 3.7) durch
$p - n$ abschätzen. Die Klassenverteilung der Beispiele in der Trainingsmenge ist dabei unerheblich. Wie in Abbildung 9 zu sehen ist, schließen die Isometrien einen $45^\circ$ Winkel mit der $x$-Achse ein. Die Isometrien sind also Geraden mit der Steigung 1. Durch Setzen der Parameter $a = b = d = 1$ oder $c = \frac{1}{2}$ erhält man aus Gleichung 3.3 die Heuristik \textit{Accuracy}.

\[
\text{Accuracy}: h_{\text{acc}}(n, p) = p - n = \frac{p + (N - n)}{P + N} \quad (3.7)
\]

Abbildung 9: Isometrien von Accuracy und Weighted Relative Accuracy

Ein Nachteil der Heuristik ist, dass es gleich gut ist ein positives Beispiel abzudecken und ein negatives Beispiel nicht abzudecken. Dieser Nachteil kommt allerdings erst zum tragen, wenn die a priori Verteilung der Beispiele nicht repräsentativ für eine Domäne ist, oder die Kosten einer falschen Vorhersage nicht bekannt sind. Um diesem Nachteil entgegenzuarbeiten ist es sinnvoll die Heuristik anhand der Trainingsmenge zu normalisieren. Das Normalisieren der Heuristik \textit{Accuracy} führt zu Weighted Relative Accuracy (WRA) [FuFl05]. WRA wird durch Gleichung 3.8 definiert.

\[
\text{WRA}: h_{\text{wra}}(n, p) = \frac{p - n}{N} = TPR - FPR \approx \frac{p + n}{P + N} \left( \frac{p}{p + n} - \frac{P}{P + N} \right) \quad (3.8)
\]

Ebenfalls in Abbildung 10 lässt sich leicht der Unterschied zwischen Accuracy und WRA feststellen. Die Isometrien von WRA sind alle parallel zur Standardverteilung, was gleichbedeutend damit ist dass alle Isometrien die Steigung $\frac{p}{N}$ haben. Die Heuristik bewertet ein Steigen der $tpr$ einer Hypothese genauso gut wie ein Absinken der $fpr$ einer Hypothese. Ein weiteres Merkmal der Weighted Relative Accuracy ist, dass sie für den Fall $P = N$ gleich zur Accuracy ist. Die WRA lässt sich durch setzen der Parameter $a = \frac{1}{P}$, $b = \frac{1}{N}$ oder $c = \frac{N}{P + N}$ oder $d = \frac{P}{N}$ aus Gleichung 3.3 gewinnen.

Abbildung 10: Isometrien Precision und Laplace
Heuristiken des $h_{pr}$ – Modells: Die grundlegende Eigenschaft dieser Heuristiken ist, dass die Isometriken symmetrisch um einen Punkt des PN-Raumes rotieren. Die Isometriken der Heuristik $h_{pr}$ (Gleichung 3.9) sind in Abbildung 10 links zu sehen. Die Heuristik misst das Verhältnis von allen abgedeckten positiven Beispielen zu allen abgedeckten Beispielen einer Regel bzw. Hypothese.

$$h_{pr}(n, p) = \frac{p}{p + n} \quad (3.9)$$

Die Heuristik Precision bewertet Regeln, die nur positive Beispiele abdecken, maximal und Regeln, die nur negative Beispiele abdecken, minimal. Dies entspricht Regeln, die sich auf der $y$-Achse (Abdeckung nur positiver Beispiele) bzw. auf der $x$-Achse (Abdeckung nur negativer Beispiele) befinden. Alle anderen Regeln werden somit besser als das Minimum bzw. schlechter als das Maximum bewertet. Dies ist auch gleichzeitig ein Nachteil der Heuristik. Betrachtet man z.B. die Regeln R1, die ein positives und kein negatives Beispiel abdeckt, und die Regel R2, die 1000 positive und kein negatives Beispiel abdeckt, erkennt man, dass beide Regeln gleich bewertet ($h_{pr}(n, p) = 1$) werden, obwohl die Regel R2 eindeutig besser ist als die Regel R1.

Um diesen Nachteil auszugleichen, kann man die Annahme treffen, dass eine Regel von vornherein schon eins, zwei oder $m$ Beispiele abdeckt. Dadurch verändert sich die Bewertung für jede einzelne Regel bzw. Hypothese dahingehend, dass Regeln, die wenig positive Beispiele abdecken schlechter bewertet werden als Regeln, die viele positive Beispiele abdecken. Dies trifft unter der Annahme zu, dass $n$ für die Regeln konstant ist. Diese Heuristiken sind Modifikationen des $h_{pr}$ – Modells. Durch diese Modifikationen ist der Punkt um den die Isometriken rotieren nicht mehr der Ursprung eines PN-Raumes, sondern ein Punkt der im negativen Bereich der $x$- bzw. $y$-Achse zu finden ist. Eine sehr bekannte Modifikation des Precision-Modells ist die Laplace-Heuristik (Gleichung 3.10). Diese Heuristik geht davon aus, dass eine Regel schon mindestens ein negatives und ein positives Beispiel abdeckt. Geht man nun von den Regeln R1 und R2 aus, wie oben im Beispiel, so sieht man, dass die Regel R2 ($h_{lap}(n, p) = 1$) besser bewertet wird als die Regel R1 ($h_{lap}(n, p) = 2/3$). Diese Annahme hat zur Folge, dass der Punkt, um den die Precision-Isometriken rotieren, nun bei (-1, -1) liegt, wie in Abbildung 10 zu erkennen ist.

$$\text{Laplace: } h_{lap}(n, p) = \frac{p + 1}{(p + 1) + (n + 1)} = \frac{p + 1}{p + n + 2} \quad (3.10)$$

Eine andere Modifikation des Precision-Modells ist das sogenannte $m$-Estimate (Gleichung 3.11). Bei dieser Heuristik ist der Ursprung der Isometriken von einem Parameter $m$ abhängig. Je nachdem wie der Parameter gewählt wird, ändern sich die Isometriken der Heuristik im PN-Raum. Generell kann man sagen, dass der Punkt $(-m * \left(1 - \frac{p}{p+N}\right), -m * \frac{p}{p+N})$ der Ursprung der Precision-Isometriken des $m$-Estimates. Geht man von einer Gleichverteilung der Beispiele in der Trainingsmenge aus ($P = N$) und wählt den Parameter $m=2$, so kann man aus dem $m$-Estimate die Laplace-Heuristik herleiten. Wählt man $m=0$ so erhält man Heuristik Precision. Abbildung 11 zeigt die Isometriken für das $m$-Estimate für verschiedene Parameter, die in dieser Arbeit verwendet wurden. Der Parameter $m=22.466$ ist ein optimaler Parameter, der in [JaFu06] gefunden wurde.
Die Laplace-Heuristik und das m-Estimate sind beides Spezialfälle des Generalized m-Estimates (Gleichung 3.12). Ein besonderes Merkmal des Generalized m-Estimates ist, dass sich die Isometriken je größer m wird immer mehr denen der Heuristik WRA ähneln. Für m = ∞ sind die Isometriken des Generalized m-Estimate und die Isometriken des Generalized Cost-Measures äquivalent und für m=0 sind die Isometriken äquivalent zu denen von Precision [FüFl05]. Denn es gilt, für a = b = 1 bzw. m = 2 und c = 1/2, dass \( h_{gm} = h_{lap} \). Ein weiterer Spezialfall des Generalized m-Estimates ist das M-Estimate, der Parameter c wird dazu gleich \( \frac{p}{p+N} \) gesetzt.

\[
m - \text{Estimate: } h_m(n,p) = \frac{p + m \cdot \frac{p}{p+N}}{p + n + m} \quad (3.11)
\]

Abbildung 11: Isometriken für das m-Estimate mit verschiedenen Parametern

\[
\text{Generalized m - Estimate: } h_{gm}(n,p) = \frac{p + m \cdot c}{p + n + m} = \frac{p + a}{(p + a) + (n + b)} \quad (3.12)
\]

Weitere Heuristiken, die dem Precision-Modell angehören sind in [FüFl05] zu finden. Das Generalized m-Estimate bzw. das m-Estimate als Spezialfall sind beide parametrisierbare Heuristiken (Definition 3.3.1).

**Definition 3.3.1 (parametrisierbare Heuristik):** Eine parametrisierbare Heuristik \( H \) wägt zwischen zwei Heuristiken, \( h_1 \) und \( h_2 \), ab. Diese Abwägung wird durch einen frei wählbaren Parameter \( p \) bewerkstelligt, sodass

- Für \( p \rightarrow 0 \): \( H \) tendiert zu \( h_1 \) oder
- Für \( p \rightarrow \infty \): \( H \) tendiert zu \( h_2 \)

gilt.
Das Generalized m-Estimate wägt zwischen Heuristiken des Precision-Modells und Heuristiken des Linear-Cost-Modells ab [FüFl05]. Eine weitere parametrisierbare Heuristik ist das Klösgen-Maß (Gleichung 3.15) [Klös92], das die Heuristiken PrecisionGain (Gleichung 3.13) und Coverage (Gleichung 3.14) abwägt. Coverage misst die Anzahl der abgedeckten Beispiele im Verhältnis zur Anzahl aller Beispiele, während PrecisionGain sich ähnlich wie Precision verhält. Der Unterschied zwischen Precision und PrecisionGain ist der zusätzliche Faktor \( \frac{p}{p+n} \), der noch die a priori Verteilung der Beispiele in der Trainingsmenge mit betrachtet. Somit wird eine Regel, die keine negativen Beispiel abdeckt im besten Fall mit \( \frac{1}{p+n} \) und eine Regel die keine positiven Beispiele abdeckt im schlechtesten Falls mit \( \frac{p}{p+n} \) bewertet. Dies soll eine zu optimistische Bewertung einer Regel bzw. Hypothese verhindern.

\[
\text{PrecisionGain: } h_{pg}(n,p) = \frac{p}{p+n} - \frac{P}{P+N} \quad (3.13)
\]
\[
\text{Coverage: } h_{cov}(n,p) = \frac{p+n}{P+N} \quad (3.14)
\]

Um die Tendenz des Klösgen-Maßes hin zu einer dieser Heuristiken festzulegen, werden beide Heuristiken mit dem Parameter \( \omega \) wie folgt verknüpft:

\[
\text{Klösgen-Maß: } h_{klösg}(n,p) = h_{cov}^\omega \ast h_{pg} = \left( \frac{p+n}{P+N} \right)^\omega \ast \left( \frac{p}{p+n} - \frac{P}{P+N} \right) \quad (3.15)
\]

Auffällig ist, dass das Klösgen-Maß eigentlich drei Heuristiken abwägt. Dazu können für den Parameter \( \omega \) zwei Intervalle bestimmt werden, in denen diese Abwägung zu erkennen ist. Im Intervall \( \omega \in [0,1] \) verhält sich die Heuristik wie Precision bzw. WRA. Für den Fall \( \omega = 0 \) ist das Verhalten äquivalent zu dem von Precision. Für \( \omega = 1 \) verhält sich das Klösgen-Maß äquivalent zur Weighted Relative Accuracy. Das zweite Intervall, das sich für \( \omega \) identifizieren lässt, liegt zwischen \( [1, \infty) \). Je größer \( \omega \) wird, desto mehr ähneln die Isometriken des Klösgen-Maßes denen der Heuristik Coverage. In Abbildung 12 ist dieses Verhalten für verschiedene Werte von \( \omega \) gezeigt. Der Wert 0.4323 für \( \omega \) ist ein optimaler Wert, der in [JaFü06] bestimmt wurde. Die Werte 0,3 und 0,2 sind zwei weitere Parameter, die in dieser Arbeit verwendet werden.

Abbildung 12: Isometriken des Klösgen-Maßes für verschiedene Werte von \( \omega \)
3.3.2 Nicht-Lineare Heuristiken

Die Heuristiken, die bis jetzt betrachtet wurden, hatten alle lineare Isometrien gemeinsam. Das Klößgen-Maß zeigte lineare Isometrien nur für zwei verschiedene Werte. Die Isometrien der nicht-linearen Heuristiken sind Kurven im PN-Raum. Die zwei Vertreter dieser Familie von Heuristiken, die hier vorgestellt werden sind Correlation [FürFl05] und FOILGain [QuCJ95].


Correlation: \( h_{corr}(n, p) = \frac{p(N - n) - n(P - p)}{\sqrt{PN(p + n)(P - p + N - n)}} = \frac{pN - np} {\sqrt{PN(p + n)(P - p + N - n)}} \) (3.16)

Die Isometrien der Heuristik (Abbildung 13) sind symmetrisch um die Standardverteilung gebogen. Regeln, die sich auf der Geraden der Standardverteilung befinden werden von der Heuristik mit 0 bewertet, wohingegen Regeln die nahe am Punkt (0, P) zu finden sind, mit positiven Werten <1 und Regeln nahe dem Punkt (N, 0) mit negativen Werten > -1 bewertet werden. Die Heuristik tendiert dazu vollständige Regeln zu lernen, d.h. Regeln die keine negativen Beispiele abdecken. Je näher eine Regel dem Punkt (0, P) ist, desto besser wird die Heuristik Regeln bewerten, die negative Beispiele ausschließen, als Regeln die mehr positive Beispiele abdecken. Ein weiteres Merkmal der Heuristik ist, dass sie äquivalent zu einem normalisierten \( \chi^2 \)-Test über der Konfusionsmatrix ist [FürFl05].

Eine weitere Heuristik mit nicht-linearen Isometrien im PN-Raum wird in FOIL [QuCJ95] verwendet. Die Heuristik, hier als FOILGain (Gleichung 3.18) bezeichnet, die FOIL verwendet ist eine Modifikation der InformationGain Heuristik [FürFl05]. FOILGain bewertet nicht eine Regel, sondern die Differenz des InformationGains (Gleichung 3.17) zwischen einer Regel \( r \) und ihrer direkten Vorgängerregel \( r^* \).
Der Parameter \( c \) in Gleichung 3.18 ist der Precision-Wert der Vorgängerregel \( r' \) bevor diese verfeinert wurde. Die Isometrien von FOILGain zeigen im PN-Raum sowohl nicht-lineares als auch lineares Verhalten. Auch verändert sich die Landschaft der Isometrien, je nachdem wie die Vorgängerregel bewertet worden ist. Der Einfluss der Vorgängerregel und die Veränderung der Isometrien sind in Abbildung 14 visualisiert. Anhand der Isometrien lässt sich eine Kurve bzw. Basis-Isometrik identifizieren [FüFl05]. Diese Basis-Isometrik beginnt im Punkt \((0, 0)\) und hat die Steigung \( \frac{c}{c-1} \). Auf der Basis-Isometrik liegende Regeln werden mit dem Wert 0 bewertet. Befindet sich eine Regel unterhalb dieser Kurve, so wird sie negativ bewertet, was einem Verlust an Information entspricht. Regeln, die oberhalb dieser Kurve liegen werden positiv bewertet und sind somit Regeln, die zur weiteren Verfeinerung oder für die Hypothese in Frage kommen.

\[
\text{InformationGain: } h(n,p) = \log_2 \frac{p}{p+n} \quad (3.17)
\]

\[
\text{FOILGain: } h_{\text{foil}}(n,p) = p \left( \log_2 \frac{p}{p+n} - \log_2 c \right) \quad (3.18)
\]
Eine Regel bzw. eine Hypothese, die auf verrauschten oder fehlerhaften Trainingsdaten gelernt wird, ist meist sehr komplex. Die Hypothese ist an die Trainingsdaten überangepasst und verallgemeinert nur schlecht für vorher ungesiehene Beispiele. Sie enthält viele Regeln, und jede einzelne Regel deckt nur wenige positive Beispiele ab. Die Regeln enthalten zusätzlich noch viele Attributtests, die der Regel nur hinzugefügt wurden um negative Beispiele auszuschließen. Neben der Wahl einer geeigneten Heuristik für das Lernen, ist das Pruning eine weitere Methode ein Overfitting einer Hypothese an verrauschte Daten zu vermeiden und eine Hypothese zu erhalten die besser auf vorher nicht gesehenen Beispielen verallgemeinert. Das Pruning selbst greift an zwei Punkten des Lernprozesses, um eine Überangepasstheit zu vermeiden.

Um das Pruning an sich etwas zu veranschaulichen dient die Abbildung 15. Die einzelnen Felder der Abbildung enthalten Hypothesen, die mit unterschiedlichen Pruningmethoden gelernt wurden. Regeln, die in der Hypothese enthalten sind, sind in Abbildung 15 weiß dargestellt. Zusätzlich zu den weißen Regeln enthält die Abbildung noch graue Regeln, die durch Postpruning-Mechanismen aus der...


Beide Mechanismen lassen sich auf zwei verschiedene Arten miteinander verbinden. Das kombinierte Pre- und Postpruning (vgl. Abbildung 15, links unten) erhält man, wenn man das Lernen mit Prepruningmechanismen unterstützt und die so gelernte Hypothese zusätzlich noch postprun. Weiter kann man beide Pruningmechanismen auch integrieren (vgl. Abbildung 15, rechts unten). Das Integrierte Pre- und Postpruning lernt eine überangepasste Regel und prun diese sofort. Dies ist der Postpruningschritt. Nach jeder Regel wird geprüft, ob die Hypothese bereits genügend Regeln enthält, d.h. verschlechtert sich die Bewertung der Hypothese, wenn die Regel mitaufgenommen wird im Vergleich zu der Hypothese ohne diese Regel. Ist dies der Fall so wird das Lernen abgebrochen [Fürn97].

Das Prepruning wird beim Lernen durch sogenannte Stopkriterien, die Entscheiden wann eine Regel bzw. Hypothese hinreichend genau ist, bewerkstelligt (siehe Kapitel 4.2.1). Im Gegensatz dazu werden beim Postpruning sogenannte Pruningoperatoren (Definition 4.1) verwendet um eine Hypothese bzw. eine Regel zu verallgemeinern.

**Definition 4.1 (Pruningoperator):** Ein Pruningoperator transformiert eine Hypothese/Regel in eine weitere, allgemeinere Hypothese/Regel oder lässt die Regel unverändert, falls keine allgemeinere Regel erzeugt werden kann.


- **delete-last-condition Operator** [BrPa91]: Der Operator löscht den letzten Attributtest einer Regel.
- **delete-rule Operator** [BrPa91]: Der Operator löscht eine ganze Regel.
- **delete-condition-sequence Operator** [Cohe95]: Dieser Operator löscht eine endliche Anzahl von Attributtests aus einer Regel.
- **find-best-replacement Operator** [WeIn91]: Der Operator sucht auf der Pruningmenge für einen Attributtest einer Regel, einen anderen Attributtest, der die Bewertung einer Regel auf der Pruningmenge maximiert.
- **find-best-simplification Operator**: Sei \( k \) die Anzahl der Attributtests der Regel \( R \), so erzeugt der Operator \( k \) neue Regeln, indem der Operator den ersten, zweiten, ..., \( k \)-ten Attributtest aus der Regel löscht. Der Operator wählt dann die Regel, aus den \( k \) Regeln, die ein bestimmtes Qualitätsmaß maximiert. Die Regel, auf die der Operator
angewendet wird, wird auch mitbetrachtet. Besitzt diese Regel schon den maximalen Wert, so wird diese Regel ausgewählt.

- **identity Operator**: Dieser Operator bildet eine Regel bzw. Regelmenge auf sich selbst ab.

In diesem Kapitel werden die vier Möglichkeiten mit verrauschten Daten umzugehen und eine überangepasste Hypothese zu vermeiden genauer vorgestellt und exemplarisch ein Vertreter der Separate-and-Conquer Algorithmen vorgestellt, der diese Methode implementiert.

### 4.1 Postpruning


**procedure** POSTPRUNING(Examples, Splitratio)

```
SPLITEXAMPLES(Splitratio, Examples, GrowingSet, PruningSet)
Theory = SEPARATEANDCONQUER(GrowingSet)
loop
  NewTheory = BESTSimplification(Theory, PruningSet)
  if ACCURACY(NewTheory, PruningSet) < ACCURACY(Theory, PruningSet)
    exit loop
  Theory = NewTheory
return(Theory)
```

**Abbildung 16: Postpruning Algorithmus [Fürn97]**

### 4.1.1 REP


4.2 Prepruning


4.2.1 Stopkriterien für das Prepruning


ist also eine Hypothese, für die Gleichung 4.1 erfüllt ist. Eine optimale Hypothese minimiert die linke Seite der Gleichung 4.1.

\[ L(H) + L(E|H) < L(E) \] (4.1)

\( L(H) \) ist die Länge der Hypothese in Bits, \( L(E) \) ist die Länge der Trainingsmenge in Bits, und \( L(E|H) \) ist die Anzahl der Bits, die benötigt werden um die Beispiele zu beschreiben, wenn die Hypothese gegeben ist. Die Hypothese wird also eher kurze Regeln und wenige Regeln enthalten.

Eine Variante des MDL-Prinzips ist in FOIL [QuCJ95], unter dem Namen Encoding Length Restriction, implementiert. Diese Variante vergleicht die Anzahl der Bits, die nötig sind eine Regel zu codieren, mit der Anzahl der Bits, die benötigt werden um die Beispiele, die von der Regel abgedeckt werden, zu codieren. Die Anzahl der Bits, die benötigt werden eine Regel zu kodieren ist in FOIL durch Gleichung 4.1 gegeben.

\[
\text{FOIL Encoding Length Restriction: } \text{len}(r) = \log_2(P + N) + \log_2\left(\frac{P + N}{p}\right) \] (4.2)

Die Gleichung 4.2 kodiert die Anzahl der Bits die nötig sind um die verbleibende Trainingsmenge und die Regel zu übertragen. In [FüFl05] wurde gezeigt, dass die in FOIL benutzte Variante eine Anomalie aufweist, falls die Anzahl der positiven Beispiele größer ist als die Anzahl der negativen Beispiele. Gilt \( P > N \) in der Trainingsmenge so liegt das Maximum der Funktion nicht einer Abdeckung von \( P \) positiven Beispielen, sondern das Maximum liegt bei \( \frac{(P+N)}{2} \). In Abbildung 17 ist die Anomalie im PN-Raum visualisiert. Das linke Bild zeigt das Verhalten der Isometriken für \( P < N \), wohingegen das Bild auf der rechten Seite den Fall \( P > N \) zeigt. Das Maximum ist hier durch die rote Linie gekennzeichnet.

Eine andere Möglichkeit die Länge einer Regel zu berechnen wurde in [Pfah95] vorgestellt und enthält einige Veränderungen gegenüber der in FOIL benutzten MDL-Formel. Die Formel ist in Gleichung 4.3 aufgeführt.

\[
\text{len}(\text{ruleset}) = n_{nc} \cdot e_{nc} + \sum_i \text{len}(r_i) \\
\text{len}(r_i) = rc_i + n_c \cdot e_c 
\] (4.3)

In der Formel ist \( n_{nc} \) die Anzahl der Beispiele, die nicht von der Hypothese abgedeckt werden, und \( e_{nc} \) die dazugehörige Entropie der Menge der nichtabgedeckten Beispiele. Weiter ist \( rc_i \) die Länge in Bits einer Regel, die wie in Gleichung 4.4 berechnet wird, und \( n_c \) die Anzahl der Beispiele die von der Regel abgedeckt werden und \( e_c \) die Entropie der abgedeckten Beispiele. Gleichung 4.4 berechnet die
mögliche Länge einer Regel in Bits, d.h. wie viele Möglichkeiten gibt es eine Regel der Länge \( L_{\text{th}_i} \) mit zusätzlichen Attributtests zu verlängern.

\[
rc_i = \log_2 \left( \frac{N_{pt}}{L_{\text{th}_i}} \right) \quad (4.4)
\]

Wenn die Länge einer Regel mit MDL berechnet wird, muss diese noch um den Faktor \( \log_2 \left( L_{\text{th}_i} \right) \) verkleinert werden, da die Reihenfolge der Attributtests in einer Regel irrelevant ist.

Das dritte und letzte Stopkriterium, das hier vorgestellt wird, ist der \textit{Signifikanztest}, der von \cite{ClNi89} im Algorithmus CN2 oder in mFOIL von \cite{DzBr92} benutzt wird. Der Signifikanztest vergleicht die Standardverteilung der positiven und negativen Beispiele in einer Trainingsmenge mit der Verteilung der Beispiele, die von einer Regel abgedeckt werden. Regeln oder Verfeinerungen einer Regel, deren Verteilung nicht signifikant von der Standardverteilung abweichen, werden nicht in die Hypothese mit aufgenommen bzw. nicht weiter verfeinert. Die \textit{likelihood ratio statistic (LRS)} \cite{ClNi89} ist annähernd wie eine Chi-Quadrat-Verteilung mit einem Freiheitsgrad verteilt und kann deswegen mit dieser verglichen werden. Ist der Wert der \( LRS \) größer als ein vorgegebener Schwellwert, so wird angenommen, dass eine Regel bzw. Regelverfeinerung signifikant ist. Die Schwellwerte können aus Tabelle 7 abgelesen werden.

\[
LRS: h_{LRS} = 2 \* \left( p \* \log_2 \frac{p}{e_p} + n \* \log_2 \frac{n}{e_n} \right) \quad (4.5)
\]

\[
e_p = (p + n) \* \frac{p}{p + N} \quad (4.6), \quad e_n = (p + n) \frac{N}{p + N} \quad (4.7)
\]

<table>
<thead>
<tr>
<th>Freiheitsgrad</th>
<th>0,005</th>
<th>0,01</th>
<th>0,025</th>
<th>0,05</th>
<th>0,1</th>
<th>0,5</th>
<th>0,9</th>
<th>0,975</th>
<th>0,99</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0,02</td>
<td>0,45</td>
<td>2,71</td>
<td>3,84</td>
<td>6,63</td>
</tr>
</tbody>
</table>

Tabelle 8: \( \chi^2 \)-Test Tabelle für einen Freiheitsgrad\cite{Wiki08}

<table>
<thead>
<tr>
<th></th>
<th>positiv klassifiziert</th>
<th>negativ klassifiziert</th>
</tr>
</thead>
<tbody>
<tr>
<td>positives Beispiel</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>negatives Beispiel</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Tabelle 9: Beispiel LRS und Chi-Quadrat

Der \( \chi^2 \) -Test errechnet mit der Konfusionsmatrix die Wahrscheinlichkeit, dass eine bestimmte Häufigkeitsverteilung einer Menge signifikant ist. Die Werte eines \( \chi^2 \)-Test lassen sich aus einer Tabelle ablesen, als Beispiel soll hier die Tabelle 8 dienen. Die Tabelle, aus \cite{Wiki08} übernommen, umfasst die Werte des \( \chi^2 \)-Test mit einem Freiheitsgrad und verschiedenen Wahrscheinlichkeiten. Der Vergleich der LRS-Heuristik mit dem \( \chi^2 \) -Test liefert die Wahrscheinlichkeit, dass die Häufigkeitsverteilung einer Regel sich signifikant von der Standardverteilung der Beispiele unterscheidet.

Als Beispiel soll die Konfusionsmatrix aus Tabelle 8 dienen. Der Chi-Quadrat-Test wird wie in Gleichung (4.8) aus der Konfusionsmatrix berechnet:
\[ \chi^2 = \sum_{i,j} \frac{(k_{i,j} - E(k_{i,j}))^2}{E(k_{i,j})} = \frac{(k_{0,0} + k_{0,1} + k_{1,0} + k_{1,1}) \times (k_{0,0} \times k_{1,1} - k_{0,1} \times k_{1,0})^2}{(k_{0,0} + k_{0,1}) \times (k_{1,1} + k_{1,0}) \times (k_{0,0} + k_{1,0}) \times (k_{1,1} + k_{0,1})} \quad (4.8) \]

Setzt man die Werte aus Tabelle 9 in die Gleichungen 4.8 ein so erhält man einen Wert von \( \chi^2 = \frac{20 \times (56 - 6)^2}{10 \times 10} = \frac{5051}{100} = 5,051 \) und für die LRS Heuristik \( h_{LRS}(n, p) = 3,85 \). Anhand der Werte kann man aus Tabelle 7 ablesen, dass die Häufigkeitsverteilung bzw. die Verteilung der positiven und negativen Beispiele der Regel mit mindestens 97,5%-iger Sicherheit von der Standardverteilung abweichen. Die Regel wird damit in die Hypothese mit aufgenommen bzw. die Regel wird weiter verfeinert und gilt damit als signifikant.


### Abbildung 18: Likelihood Ratio Statistic Isometriken

#### 4.3 Combining Post- & Prepruning


In [Cohe93] wurde ein solcher Ansatz vorgestellt. Die Lernphase von Grow wurde durch zusätzliche Stopkriterien, die auf dem MDL-Prinzip basieren, beschleunigt. Das Prepruning soll die Überanpassung an die Trainingsdaten nicht vollständig vermeiden, sondern nur reduzieren, damit die gelernte Hypothese weniger komplex ist. Ein Nachteil bei dieser Herangehensweise ist, dass das
Prepruning eine Hypothese finden kann, die in der Postpruningphase nicht verwendet werden kann, da sie zu simpel ist.


4.4 Integrating Post- & Prepruning
Eine andere Art beide Pruningmethoden zusammenzuführen, bietet das integrierte Post- und Prepruning. Die Idee dahinter ist, dass nicht als erstes eine konsistente und vollständige Hypothese auf den Trainingsdaten gelernt wird und somit möglicherweise auch unnötige Attributtests einer Regel hinzugefügt werden, sondern dass zuerst eine vollständige und konsistente Regel gelernt und direkt geprüft wird. Dies soll dem Lerner ermöglichen die Beispiele zu entfernen, die von einer Regel abgedeckt werden, nachdem diese geprüft wurde, und so das Hinzufügen eines unnötigen Attributtests zu einer Regel zu vermeiden. Das sogenannte Inkrementelle Reduced Error Pruning (IREP) [FuWi94] implementiert diese Strategie. Der darauf basierende Algorithmus RIPPER [Cohe95] erweitert den Algorithmus so dass die Fehlerraten des Algorithmus gesenkt werden. Eine Implementierung des RIPPER, namens JRIP [WFTH99], wird in Kapitel 4.4.2 genauer vorgestellt.

4.4.1 IREP – Incremental Reduced Error Pruning

Abbildung 19 ist der Algorithmus IREP in Pseudocode gegeben, wie er auch in [Fürn97] zu finden ist. Die Trainingsmenge wird als erstes wieder in zwei kleinere Mengen aufgeteilt, die Growing- und Pruningmenge. Im Unterschied zum REP geschieht dies aber nicht nur bevor das Lernen einer Hypothese beginnt, sondern immer vor dem Lernen einer Regel. Nachdem eine Regel gelernt wurde, wird diese direkt geprüft (BestSimplification) und versucht die Genauigkeit der Regel auf der Pruningmenge zu erhöhen. Im ursprünglichen Algorithmus wird der find-best-simplification-Operator
auf eine Regel angewendet, aber andere oder mehrere Pruningoperatoren sind hier durchaus vorstellbar. Dies wird solange gemacht, bis eine geprunte Regel eine schlechtere Genauigkeit besitzt als ihre direkte Vorgängerregel. Die zweite *if*-Abfrage soll gewährleisten, dass die gefundene Regel eine Mindestgenauigkeit auf der Pruningmenge hat bzw. mehr positive als negative Beispiele abdeckt. 

Ist eine Regel gefunden worden und zur Hypothese hinzugefügt worden, so werden die Beispiele, die von der Regel abgedeckt werden, aus der Trainingsmenge entfernt. Die Beispiele werden nicht nur aus der Growing-Menge entfernt, sondern auch aus der Pruning-Menge. Bei der nächsten Iteration werden die Beispiele erneut in zwei Mengen geteilt und die Suche nach neuen Regeln beginnt von vorne.

Der Algorithmus besitzt Eigenschaften, die die Probleme von REP lösen, bringt aber auf der anderen Seite neue Probleme mit. Das Effizienzproblem von REP ist eindeutig reduziert worden durch die Art wie eine Hypothese bei IREP gelernt wird. Dadurch, dass die Regeln einer Hypothese gelernt werden und gleich danach geprunt werden, ist die Zeitkomplexität von IREP für den *delete-last-condition* Operator proportional zu $O(n \cdot \log^2(n))$ [Fürn97]. Die Separate-and-Conquer Strategie ist für diesen Algorithmus kein Problem mehr, da die Beispiele erst aus der Trainingsmenge entfernt werden, nachdem eine Regel komplett fertig gelernt und geprunt wurde. Die Herangehensweise wie eine Hypothese gelernt wird, ist hier ähnlich zu der in Grow benutzten Weise mit dem Unterschied, dass nicht als erstes eine überangepasste Hypothese gelernt werden muss, sondern diese direkt aus einzelnen gelernten und geprunten Regel zusammengesetzt wird. Somit ist der Algorithmus auch weniger Anfällig für lokale Optima [Fürn97]. Auch das Problem mit der Aufteilung der Trainingsmenge in Growing- und Pruningmenge ist hier zumindest reduziert. Das Problem beschränkt sich hier nur noch auf einzelne Regeln und nicht mehr auf eine komplette Hypothese.

4.4.2 RIPPER und JRIP


RIPPER lernt Regel mit dem FOIL Algorithmus. Die Suchheuristik, die FOIL verwendet ist in Abbildung 19 visualisiert und durch die Formel aus Gleichung 3.19 gegeben. Die durch FOIL gelernten Regeln, werden dann auf der Pruningmenge vereinfacht. IREP verwendet zum Vereinfachen der gelernten Regel den *delete-last-condition* Operator, der in jedem Schritt ein Attributtest aus der Regel löscht und zwar so, dass die Heuristiken $\frac{P}{P+N}$ (IREP-2) bzw. $\frac{P+(N-n)}{P+N}$ (IREP) einen bestimmten Wert nicht unterschreiten. In RIPPER ist zum Prunen ein anderes Maß implementiert worden, das in Gleichung (4.9) zu sehen ist. Die Heuristik, die vom RIPPER zum Prunen benutzt wird ist äquivalent
zu Heuristik Precision [FüFl05]. Demensprechend sind die Isometriken der Heuristik aus Gleichung 4.9 äquivalent zu denen aus Abbildung 10 (vgl. Kapitel 3.3.1).

\[ RIPPER: h_{RIPPER}(n, p) = \frac{p-n}{p+n} \] (4.9)

Der Algorithmus RIPPER löscht während dem Prunen nicht nur immer einen Attributtest einer Regel, sondern eine endliche Anzahl von Attributtest, sodass die Heuristik aus Gleichung 4.9 maximiert wird. Diese Modifikation bilden die Grundlage des IREP*-Algorithmus [Cohe95], der in RIPPER Verwendung findet. Die so gelernte Hypothese wird in RIPPER zusätzlich noch einer Optimierungsphase übergeben.

Die Optimierungsphase soll die gefundene Hypothese noch weiter verbessern. Dies geschieht nach einem einfachen REP-Prinzip. Die Hypothese, die während der ersten Phase gelernt wurde, wird Regel für Regel verbessert. Die Regeln aus der Hypothese werden in der Reihenfolge, in der sie gelernt wurden betrachtet. Für eine Regel \( R_i \) aus der Hypothese \( \{R_1, R_2, ..., R_i, ..., R_n\} \) werden zwei neue Regeln erzeugt. Die erste erzeugte Regel wird Replacement von \( R_i \) genannt. Das Replacement wird auf der Growingmenge gelernt, sodass der Fehler der Hypothese ohne Replacement aber mit dem Replacement verringert wird. Die zweite Regel, die aus der Regel \( R_i \) erzeugt wird, wird Revision von \( R_i \) genannt. Die Revision von \( R_i \) wird auf der Growingmenge weiter verfeinert, indem weitere Attributtests zu \( R_i \) hinzugefügt werden. Nachdem das Replacement und die Revision von \( R_i \) erzeugt wurden, wird anhand des MDL-Prinzips entschieden, welche Regel in der endgültigen Hypothese enthalten ist. Also wird entweder \( R_i \), das Replacement oder die Revision wieder in die Hypothese aufgenommen. Die Optimierungsphase soll die Effekte des konventionellen REP annähern, sodass die Genauigkeit der Hypothese noch erhöht wird.

Kapitel 5 Technische Umsetzung im SeCo-Framework


5.1 Das SeCo-Framework


<table>
<thead>
<tr>
<th>Paketname</th>
<th>Autor</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>seco.heuristics</td>
<td>JF</td>
<td>Dieses Paket enthält die Klassen für die Suchheuristiken</td>
</tr>
<tr>
<td>seco.learners</td>
<td>JF</td>
<td>Dieses Paket enthält die verschiedenen Regellernalgorithmen</td>
</tr>
<tr>
<td>seco.model</td>
<td>JF</td>
<td>Dieses Paket enthält die Klassen für das Modell der Hypothesensprache</td>
</tr>
</tbody>
</table>

Tabelle 10: SeCo-Framework Pakete

Methoden, die das Overfitting vermeiden, sind in diesem Framework nur an wenigen Stellen implementiert. Die wichtigsten Methoden, die implementiert sind, sind die Gewährleistung, dass das Hinzufügen einer Regel zu einer Regelmenge die Genauigkeit nicht verringert. Diese Methode ist in der Klasse Covering implementiert und wird im weiteren Verlauf als $\text{IncAcc}^6$ bezeichnet. Die zweite Methode ist im Suchalgorithmus implementiert und wird als $\text{ForwardPruning}$ bezeichnet. Das $\text{ForwardPruning}$ basiert dabei auf folgender Annahme:

$^6$ Zusammengesetzt aus „increasing“ und „accuracy“.
Vergleich von Pruningalgorithmen für Regellerner - Technische Umsetzung im SeCo-Framework

Deckt eine Regel $R$ bereits $p$ positive und $n$ negative Beispiele ab und wird von einer Heuristik mit dem Wert $h$ bewertet, so wird eine neue Regel $R_p$ erzeugt, die $p$ positive und keine negativen Beispiele abdeckt. Ist die Bewertung der Regel $R_p$ schlechter als die Bewertung der momentan besten Regel im Strahl, so wird das Regellernen unterbrochen, da keine Verfeinerung der Regel $R_1$ eine besser Bewertung erhalten kann als die Bewertung der momentan besten Regel.

5.1.1 Ein generischer Pruningalgorithmus

```
procedure PruningSeparateAndConquer(Examples)

Theory = ∅
OUTERSPLIT (Examples, Splitratio, GrowingSet, PruningSet)
while POSITIVE(GrowingSet) ≠ ∅
    Rule = FindBestRule(GrowingSet)
    Covered = COVER(Rule, GrowingSet)
    Rule = POSTPROCESSRULE(Rule, PruningSet)
    if RULESTOPPINGCRITERION(Theory, Rule, GrowingSet, PruningSet)
        exit while
    Examples = Examples \ Covered
    INNERSPLIT (Examples, Splitratio, GrowingSet, PruningSet)
    Theory = Theory ∪ Rule
Theory = POSTPROCESS(Theory, GrowingSet, PruningSet)
return(Theory)
```

Abbildung 20: Ein generischer SeCo-Pruning-Algorithmus


5.2 Realisierung des Prunings


In Abbildung 21 sind die Pakete des SeCo-Frameworks (orange) und die Pakete, die das Framework mit Pruningfunktionalität (gelb) erweitern, zu sehen. Das Paket seco.pruning enthält die Klassen, die die Pruningalgorithmen implementieren. Die Pakete seco.pruning.operator und seco.pruning.criterion enthalten die Klassen für die Pruningoperatoren bzw. die Klassen für die Stopkriterien. Im Paket seco.pruning.model sind die abstrakten Klassen bzw. die Interfaces enthalten, die von den anderen drei Paketen benötigt werden. Die nachfolgenden Abschnitte beschreiben die Umsetzung des generischen Algorithmus aus Abbildung 20 und die Integration der Klassen in das Framework.
5.2.1 Pruningklassen

```java
public abstract class PruningTemplate{
    
    public abstract Instances[] outerSplit(Instances);
    public abstract Instances[] innerSplit(Instances);
    public abstract CandidateRule postProcessRule(CandidateRule, Instances);
    public abstract RuleSet postProcess(RuleSet, Instances);
    // Methode für das StoppingCriterion
    public abstract boolean literalStop(CandidateRule, Instances);
    // Methode für das RuleStoppingCriterion
    public abstract boolean clauseStop(RuleSet, CandidateRule, Instances);
    
}
```

Abbildung 22: Klasse PruningTemplate


5.2.2 Pruningoperatoren


Die Operatoren, die im Verlauf der Arbeit implementiert wurden, sind delete-rule, delete-last-condition, indentity und find-best-simplification. Das Verhalten der Operatoren, entspricht dem Verhalten der Operatoren, die in Kapitel 4 beschrieben sind.
5.2.3 Stopkriterien
Obwohl für jeden Pruningalgorithmus ein eigenes Stopkriterium implementiert werden kann, ist es
dennoch sinnvoll die Methoden literalStop und clauseStop nicht nur als überschreibbare Methoden zu
betrachten, sondern auch als Delegationsmethoden. Dies ist deswegen von Vorteil, da so ein
Algorithmus nicht nur auf ein bestimmtes Stopkriterium festgelegt ist, sondern derselbe Algorithmus
auch mit unterschiedlichen Stopkriterien implementiert werden kann. Aus diesem Grund wurde bei
den Stopkriterien dasselbe Entwurfsmuster, das auch bei den Pruningoperatoren verwendet wird,
zugrunde gelegt. Ob das Stopkriterium nun als RuleStoppingCriterion oder als StoppingCriterion
fungiert, wird durch zwei Interfaces festgelegt, die die entsprechende Methode zur Verfügung stellen.
In Abbildung 24 sind die beiden Interfaces kurz im Pseudocode dargestellt. Für die Stopkriterien
existiert auch eine abstrakte Klasse. Diese dient lediglich dazu einige Parameter, wie einen
Schwellwert oder eine Heuristik für ein Stopkriterium zu konfigurieren.

```
public abstract class RuleSetOperator{
    :

    public Vector<RuleSet> applyOperator(RuleSet, Instances){
        Vector<RuleSet> sets = new Vector<RuleSet>;
        for each Rule R in RuleSet{
            rs = RuleSet \ R;
            r' = prune(R);
            rs = rs \ r';
            sets.add(rs);
        }
        return sets;
    }

    protected abstract CandidateRule prune(CandidateRule, Instances);
    :
}

public abstract class RuleOperator{
    :

    public CandidateRule applyOperator(CandidateRule, Instances){
        r' = pruneRule(R);
        return r';
    }

    protected abstract CandidateRule prune(CandidateRule, Instances);
    :
}
```

Abbildung 23: Klasse RuleSet- und RuleOperator

Als Stopkriterien wurden in dieser Arbeit fünf verschiedene Kriterien implementiert. Das erste
Stopkriterium, das implementiert wurde, ist das Kriterium, das in IREP [FüWi94] verwendet wurde.
Das Kriterium MaximumErrorRate stoppt Regelverfeinerungen bzw. das Hinzufügen von Regeln,
wenn der Fehler der gelernten Regeln auf der Pruningmenge 50% übersteigt. Das CutOff-Kriterium
wurde gleich auf zwei Arten implementiert. Die erste stoppt das Regellernen, wenn die Bewertung einer Regel einen bestimmten Schwellwert übersteigt. Da die Bewertung der Regel auf den Regeleigenschaften einer einzigen Regel berechnet wird, d.h. den momentan abgedeckten \( tp \), \( fp \) etc., handelt es sich um eine absolute Bewertung einer Regel. Dieses Kriterium wird im weiteren Verlauf der Arbeit als absolutes CutOff-Kriterium bezeichnet. Das zweite CutOff-Kriterium vergleicht die gelernte Regel mit ihrer Vorgängerregel. Ist die Bewertung dieses Vergleichs größer als ein bestimmter Schwellwert, so wird das Kriterium als wahr ausgewertet und bricht das Lernen ab. Da die Bewertung einer Regel bei diesem Kriterium relativ zu ihrer Vorgängerregel gemacht wird, wird dieses Kriterium im weiteren Verlauf als relatives CutOff-Kriterium bezeichnet. Die letzten zwei Kriterien, die implementiert wurden, sind das Significance Testing und ein auf dem MDL-Prinzip (siehe Kapitel 4.2.1) basierendes Kriterium.

Aufgrund einer Besonderheit des Covering-Algorithmus in Verbindung mit Stopkriterien, die eine Regel als signifikant oder gut bewerten, obwohl die Regel noch mehr negative als positive Beispiele abdeckt, ist es notwendig die Stopkriterien mit einer zusätzlichen Bedingung zu versehen. Diese zusätzliche Bedingung soll garantieren, dass eine Regel immer mehr positive als negative Beispiele abdeckt. Ohne diese zusätzliche Bedingung kann es leicht dazu kommen, dass nur leere Hypothesen gelernt werden, da einige der implementierten Stopkriterien im Konflikt mit dem IncAcc-Kriterium, das im Covering-Algorithmus enthalten ist, stehen.

Das Significance Testing in der vorliegenden Implementierung überprüft in diesem Sinn, mit welcher Sicherheit die Anzahl der positiven und negativen Beispiele, die von einer Regel abdeckt werden, von der Standardverteilung in Richtung des Punktes \((0, P)\) im PN-Raum abweicht.

### 5.2.4 Konfiguration der Komponenten


\[-H \text{"seco.heuristics.MEstimate -M 0.5"} \quad (5.1)\]
\[-D \quad (5.2)\]

Die Option aus Gleichung 5.1 legt die Suchheuristik, in diesem Falle das m-Estimate, für einen Lernalgorithmus fest. Die nachfolgende Option \(-M 0.5\) konfiguriert das m-Estimate so, dass der Parameter \( m \) gleich 0.5 gesetzt wird. Gleichung 5.2 ist ein Beispiel für eine Option, die keinen weiteren Wert mehr benötigt. Ist diese Option vorhanden so wird ein interner Wert auf \textit{true} gesetzt, ansonsten auf \textit{false}.


5.3 Instanziierung des Preprunings

```java
class Prepruning extends PruningTemplate{
    public boolean literalStop(CandidateRule, Instances){
        return (<IStoppingCriterion>.literalStop(CandidateRule, Instances));
    }

    public boolean clauseStop(RuleSet, CandidateRule, Instances){
        return (<IRuleStoppingCriterion>.clauseStop(RuleSet, CandidateRule, Instances));
    }
}
```

Abbildung 25: Instanzierte Methoden für das Prepruning


5.4 Instanziierung des IREP

Eine Regel wird von IREP auf einer Growing-Menge gelernt und auf der Pruning-Menge nach dem Lernen wieder vereinfacht. Beide Mengen werden nach jedem Separate-Schritt wieder neu erstellt. Die Anzahl der Beispiele, die in einer der Menge enthalten sind, wird durch die Anzahl der Teile, in die die Trainingsmenge geteilt wird, festgelegt. Sei `fold` die Anzahl der Teile, in die die Trainingsmenge aufgeteilt wird, dann ist die Anzahl der Beispiele in der Growing-Menge wie folgt festgelegt:

\[
\text{Anzahl der Beispiele in der Trainingsmenge (5.3)} = \frac{folds - 1}{folds} \times \text{Anzahl der Beispiele in der Trainingsmenge (5.3)}
\]


Aufgrund der Aufteilung der Trainingsmenge in Growing- und Pruningmenge und den im Covering-Algorithmus bzw. im IREP enthaltenen Pruningmechanismen, lernt IREP häufig leere Hypothesen oder für eine Klasse keine Regeln. Dies geschieht, wenn eine Regel soweit geprunen wird, sodass die Regel keine Attributtest mehr enthält, d.h. die Regel leer ist. Leere Regeln werden nicht in die Hypothese mit aufgenommen. Ein zweiter Grund, warum dies passiert, ist, dass das Prunen dazu


\begin{verbatim}
class IREPruning extends PruningTemplate{
    public Instances[] outerSplit(Instances){
        data[0] = draw randomly \(\frac{folds-1}{folds} \times \text{size(Instances)}\) examples from Instances
        data[1] = remaining Examples in Instances
        return(data);
    }

    public Instances[] innerSplit(Instances){
        return(outerSplit(Instances));
    }

    public CandidateRule postProcessRule(CandidateRule, Instances){
        best = CandidateRule
        do{
            pruned = RuleOperator.applyOperator(best,Instances)
            if(pruned is not better than best)
                break;
            best = pruned;
        }while(true);
        return(best);
    }

    public boolean clauseStop(RuleSet, CandidateRule, Instances){
        return(<IRuleStoppingCriterion>.clauseStop(RuleSet, CandidateRule, Instances));
    }
}
\end{verbatim}

Abbildung 26: Instanzierte Methoden für IREP
5.5 Instanziierung des Postpruning, REP

```java
class REPruning extends PruningTemplate{
    public Instances[] outerSplit(Instances){
        data[0] = draw randomly \( \frac{folds-1}{folds} \times \text{size(Instances)} \) examples from Instances
        data[1] = remaining Examples in Instances
        return(data);
    }

    public RuleSet postProcess(RuleSet, Instances){
        do{
            best = findBestSimplification(RuleSet, Instances)
            if(accuracy(best, Instances) < accuracy(RuleSet, Instances) || best.equals(RuleSet))
                break;
            RuleSet = best
        }while(true);
        return(RuleSet);
    }

    protected RuleSet findBestSimplification(RuleSet, Instances){
        curbest = RuleSet
        for each operator p
            possibleTheories = p.applyOperator(RuleSet, Instances);
        for each theory t in possibleTheories
            if(t is better than curbest)
                curbest = t
        return(curbest);
    }
}
```

Abbildung 27: Instanzierte Methoden für REP

5.6 Instanziierung des IREPOpt
Die Klasse IREPOpt, die einen Algorithmus, der an den RIPPER angelehnt ist, implementiert, ist eine Erweiterung der IREP Klasse. Zusätzlich zu dem eigentlichen IREP Algorithmus ist die Klasse so erweitert worden, dass die gelernte Hypothese nochmals eine Optimierungsphase durchläuft. Das verwendete Stopkriterium, der Pruningoperator und die Pruningheuristik sind in dieser Klasse fest

```java
class IREPOpt extends PruningTemplate{
  public RuleSet postProcess(RuleSet, Instances){
    RuleSet best;
    for each rule r in RuleSet
      revision = learn new rule for class r.class on Instances
      replacement = refine r on Instances
      newrule = the best rule from r, revision, replacement
      best.add(newrule)
    return(best);
  }

class CandidateRule postProcessRule(CandidateRule, Instances){
  best = CandidateRule
  do{
    pruned = deleteLastCondition(best)
    if(Evaluation(pruned) < Evaluation(best))
      break;
    best = pruned;
  }while(true);
  return(best);
}
```

Abbildung 28: Instanzierte Methoden für IREPOpt
Kapitel 6 Algorithmen im Vergleich

6.1 Beschreibung der Testdatensätze

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Name des Datensatz</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>anneal</td>
<td>798</td>
<td>32s 6n</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>audiology</td>
<td>226</td>
<td>69s</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>auto-mpg</td>
<td>398</td>
<td>3s 5n</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>autos</td>
<td>205</td>
<td>10s 16n</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>balance-scale</td>
<td>625</td>
<td>1s 4n</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>balloons</td>
<td>76</td>
<td>5s</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>breast-cancer</td>
<td>286</td>
<td>10s</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>breast-w</td>
<td>699</td>
<td>1s 9n</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>breast-w-d</td>
<td>699</td>
<td>10s</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>bridges2</td>
<td>108</td>
<td>11s 1n</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>cleveland-heart-disease</td>
<td>303</td>
<td>8s 6n</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>colic</td>
<td>368</td>
<td>16s 7n</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>colic.ORIG</td>
<td>368</td>
<td>20s 7n</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>contact-lenses</td>
<td>24</td>
<td>5s</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>credit</td>
<td>490</td>
<td>10s 6n</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>credit-a</td>
<td>690</td>
<td>10s 6n</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>credit-g</td>
<td>1000</td>
<td>14s 7n</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>diabetes</td>
<td>768</td>
<td>1s 8n</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>echocardiogram</td>
<td>132</td>
<td>2s 7n</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>flag</td>
<td>194</td>
<td>18s 10n</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>glass</td>
<td>214</td>
<td>1s 8n</td>
<td>7</td>
</tr>
<tr>
<td>22</td>
<td>glass2</td>
<td>163</td>
<td>1s 8n</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>hayes-roth</td>
<td>132</td>
<td>5s</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>heart-c</td>
<td>303</td>
<td>7s 6n</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>heart-h</td>
<td>294</td>
<td>7s 6n</td>
<td>5</td>
</tr>
<tr>
<td>26</td>
<td>heart-statlog</td>
<td>270</td>
<td>1s 13n</td>
<td>2</td>
</tr>
</tbody>
</table>

Tabelle 12: Teil 1 der verwendeten Datensätze
| Nr. | Datensatz       | |E|  | |A|  | |K|   |
|-----|----------------|---|---|---|---|---|
| 27  | hepatitis      | 155| 15s 6n | 2 |
| 28  | horse-colic    | 368| 15s 7n | 2 |
| 29  | house-votes-84 | 435| 17s    | 2 |
| 30  | hypothyroid    | 3163| 19s 7n | 2 |
| 31  | ionosphere     | 351| 1s 34n | 2 |
| 32  | iris           | 150| 1s 4n  | 3 |
| 33  | krkp           | 3196| 37s    | 2 |
| 34  | labor          | 57 | 10s 7n | 2 |
| 35  | labor-d        | 57 | 17s    | 2 |
| 36  | lymph          | 148| 16s 3n | 4 |
| 37  | lymphography   | 148| 16s 3n | 4 |
| 38  | machine        | 209| 1s 7n  | 8 |
| 39  | monk1          | 124| 7s     | 2 |
| 40  | monk2          | 169| 7s     | 2 |
| 41  | monk3          | 122| 7s     | 2 |
| 42  | mushroom       | 8124| 23s    | 2 |
| 43  | primary-tumor  | 339| 18s    | 22|
| 44  | promoters      | 106| 58s    | 2 |
| 45  | segment        | 2310| 1s 19n | 7 |
| 46  | sick-euthyroid | 3163| 19s 7n | 2 |
| 47  | solar-flare    | 333| 11s    | 7 |
| 48  | sonar          | 208| 1s 60n | 2 |
| 49  | soybean        | 683| 36s    | 19|
| 50  | tic-tac-toe    | 958| 20s    | 2 |
| 51  | titanic        | 2201| 4s     | 2 |
| 52  | vehicle        | 846| 1s 19n | 4 |
| 53  | vote           | 435| 16s    | 2 |
| 54  | vote-1         | 435| 16s    | 2 |
| 55  | vowel          | 990| 1s 9n  | 11|
| 56  | wine           | 178| 1s 13n | 3 |
| 57  | zoo            | 101| 17s 1n | 7 |

Tabelle 13: Teil 2 der verwendeten Datensätze

6.2 Verwendete Algorithmen

<table>
<thead>
<tr>
<th>Konfigurationsname</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRIP</td>
<td>Der RIPPER-Algorithmus mit Pruningphase und mit Optimierungsphase</td>
</tr>
<tr>
<td>JRIP-P</td>
<td>Der RIPPER-Algorithmus ohne Pruningphase, aber mit Optimierungsphase</td>
</tr>
<tr>
<td>JRIP-O</td>
<td>Der RIPPER-Algorithmus mit Pruningphase, aber ohne Optimierungsphase</td>
</tr>
<tr>
<td>JRIP-OP</td>
<td>Der RIPPER-Algorithmus ohne Pruningphase und ohne Optimierungsphase</td>
</tr>
</tbody>
</table>

Tabelle 14: JRip Konfigurationen


Der Covering Algorithmus sucht mittels einer Top-Down Hill-Climbing Suche nach Regeln. Die Suchheuristik, die während der Suche verwendet wird, kann für den Algorithmus frei gewählt werden.

6.2.1 Konfiguration von JRip und Covering
Die Algorithmen JRip und Covering bilden das Grundgerüst des Vergleichs. Die Genauigkeiten, Hypothesengrößen und Zeiten um eine Hypothese zu finden, die von diesen Algorithmen erzeugt werden, dienen dabei als grundlegende Vergleichsparameter.


6.2.2 Konfiguration der Prepruningalgorithmen

Das CutOff-Kriterium benutzt für die Entscheidung das Lernen zu unterbrechen die gleiche Heuristik, die zum Lernen verwendet wird. Zusätzlich dazu werden für das CutOff-Kriterium drei verschiedene Schwellwerte getestet. Die Cutoff - Parameter, die hier benutzt werden sind 0.3, 0.6 und 0.9. Diese Werte ergeben sich dadurch, dass die verwendeten Heuristiken Regeln im Intervall [0,1] bewerten.

Das CutOff-Kriterium benutzt für die Entscheidung das Lernen zu unterbrechen die gleiche Heuristik, die zum Lernen verwendet wird. Zusätzlich dazu werden für das CutOff-Kriterium drei verschiedene Schwellwerte getestet. Die Cutoff - Parameter, die hier benutzt werden sind 0.3, 0.6 und 0.9. Diese Werte ergeben sich dadurch, dass die verwendeten Heuristiken Regeln im Intervall [0,1] bewerten.

<table>
<thead>
<tr>
<th>Konfigurationshitename</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplace</td>
<td>Covering mit Suchheuristik Laplace</td>
</tr>
<tr>
<td>Precision</td>
<td>Covering mit Suchheuristik Precision</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Covering mit Suchheuristik Accuracy</td>
</tr>
<tr>
<td>WRAcc</td>
<td>Covering mit Suchheuristik Weighted Relative Accuracy</td>
</tr>
<tr>
<td>Klösgen</td>
<td>Covering mit Suchheuristik Klösgen-Maß. Der Parameter (\omega) hat den Wert 0.4323 (optimaler Wert, siehe [JaFü06])</td>
</tr>
<tr>
<td>Klösgen0.3</td>
<td>Covering mit Suchheuristik Klösgen-Maß. Der Parameter (\omega) hat den Wert 0.3.</td>
</tr>
<tr>
<td>Klösgen0.2</td>
<td>Covering mit Suchheuristik Klösgen-Maß. Der Parameter (\omega) hat den Wert 0.2.</td>
</tr>
<tr>
<td>MEstimate</td>
<td>Covering mit Suchheuristik m-Estimate. Der Parameter (m) hat den Wert 22.466 (optimaler Wert, siehe [JaFü06]).</td>
</tr>
<tr>
<td>MEstimate0.5</td>
<td>Covering mit Suchheuristik m-Estimate. Der Parameter (m) hat den Wert 0.5.</td>
</tr>
<tr>
<td>MEstimate13.97</td>
<td>Covering mit Suchheuristik m-Estimate. Der Parameter (m) hat den Wert 13.97.</td>
</tr>
<tr>
<td>Correlation</td>
<td>Covering mit Suchheuristik Correlation.</td>
</tr>
</tbody>
</table>

Das CutOff-Kriterium benutzt für die Entscheidung das Lernen zu unterbrechen die gleiche Heuristik, die zum Lernen verwendet wird. Zusätzlich dazu werden für das CutOff-Kriterium drei verschiedene Schwellwerte getestet. Die Cutoff - Parameter, die hier benutzt werden sind 0.3, 0.6 und 0.9. Diese Werte ergeben sich dadurch, dass die verwendeten Heuristiken Regeln im Intervall [0,1] bewerten. Der Wert 0.3 wurde gewählt, da er für das CutOff-Kriterium in Verbindung mit Correlation ein guter Wert ist [Fürn94b], und um zu sehen, ob dieser Wert auch für andere Heuristiken gut geeignet ist. Die Werte 0.6 und 0.9 wurden gewählt, um noch zwei weitere Werte innerhalb des Intervalls [0,1] zu haben und das Verhalten des CutOff-Kriteriums für größere Parameter zu untersuchen. Die Vermutung ist, dass das Verhalten gleich oder ähnlich zu dem Verhalten des CutOff-Kriteriums in Verbindung mit Correlation ist. D.h., dass für niedrige Cutoff - Parameter der Covering-Algorithmus dazu neigt, eine überangepasste Hypothese zu lernen, wohingegen für hohe Cutoff - Parameter eine leere Hypothese gelernt wird. Für das Prepruning ergeben sich somit 77 verschiedene Konfigurationen, 11 für das MDL-Kriterium, sowie jeweils 33 für das CutOff-Kriterium und Significance Testing.

<table>
<thead>
<tr>
<th>Datensatz</th>
<th>Genauigkeiten in %</th>
<th>Differenz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CutOff mit Test fp&lt;tp</td>
<td>CutOff ohne Test fp&lt;tp</td>
</tr>
<tr>
<td>contact-lenses</td>
<td>79,17</td>
<td>58,33</td>
</tr>
<tr>
<td>monk1</td>
<td>93,55</td>
<td>86,30</td>
</tr>
<tr>
<td>krkp</td>
<td>98,75</td>
<td>92,68</td>
</tr>
<tr>
<td>iris</td>
<td>91,33</td>
<td>90,67</td>
</tr>
<tr>
<td>echocardiogram</td>
<td>71,62</td>
<td>74,32</td>
</tr>
</tbody>
</table>

Tabelle 16: Genauigkeiten Prepruning mit CutOff Kriterium. Mit und ohne Test auf fp<tp.

\(^7\) Hier kurz MDL-Kriterium genannt.

Die zweite Eigenschaft, die geprüft wird, ist der eigentliche CutOff, der auf einer absoluten Bewertung der Regel basiert, wie in Kapitel 5.2.3 beschrieben. Der relative CutOff wird hier nicht betrachtet, da für die meisten verwendeten Heuristiken und gelernten Hypothesen gleiche Genauigkeiten beim Lernen zu erwarten sind oder die Differenz minimal abweicht.

Im nachfolgenden wird die Bezeichnung \textit{<Heuristik>-<Stopkriterium>-<Parameter>} verwendet, um eine Prepruningkonfiguration zu beschreiben. Die Bezeichnung \textit{<Heuristik>} orientiert sich an den Bezeichnungen aus Tabelle 15. Die Stopkriterien erhalten folgende Abkürzungen. Mit „MDL“ ist das MDL-Kriterium, mit „Sig“ das Significance Testing und mit „Cut“ das CutOff-Kriterium gemeint. Die Bezeichnung \textit{<Parameter>} kann die Werte 0.3, 0.6 und 0.9 für das CutOff-Kriterium und 90, 95 bzw. 99 für das Significance Testing annehmen. Die Bezeichnung Correlation-MDL bezieht sich dann auf die Prepruningkonfiguration, die mit der Heuristik Correlation lernt und das MDL-Kriterium als Stopkriterium verwendet, wohingegen die Bezeichnung Laplace-Sig90 die Prepruningkonfiguration, die mit Laplace lernt und das Significance Testing mit einer Sicherheit von 90% bezeichnet. In den nachfolgenden Tabellen können diese Bezeichnungen aus Platzgründen nicht mitgeführt werden.

### 6.2.3 Konfiguration der IREP Algorithmen


6.3 Evaluation der Algorithmen


1. Welche Genauigkeiten erzielen die Algorithmen? Wie groß ist die gelernte Hypothese? Wie lange braucht ein Algorithmus um die Hypothese zu lernen?
2. Welche Heuristiken erzielen die besten Ergebnisse? In welcher Konfiguration werden von den Heuristiken die besten Ergebnisse erzielt?

¹⁸ Von Prof. Fürnkranz eingeführter Begriff. Eine Gain-Heuristik bewertet eine Regel im Verhältnis zu ihrer Vorgängerregel, d.h. wie gut die Verfeinerung einer Regel ist. Im Gegensatz dazu stehen die Value-Heuristiken, die die Bewertung einer Regel nur auf diese Regel stützen, d.h. wie gut die aktuelle Regel ist.
⁹ Siehe Fußnote 7.
¹⁰ Um dies zu verdeutlichen hier eine Rechnung. Die Tabelle mit allen Daten ist 268 (Konfigurationen) * 4 (Meßwerte) * 171 (Datensätze) = 183312 Zellen groß. Wohingegen die Win-Tie-Loss Tabelle nur knapp 72000 Zellen umfasst.
¹¹ Die WTL-Tabellen befinden sich auf der beiliegenden CD, da alle Tabellen bzw. die gesamte WTL-Tabelle 574 Seiten groß ist. Verweise auf die Tabellen sind im WTL-Verzeichnis im Anhang A zu finden.
3. Wie verändern sich die Ergebnisse des Covering-Algorithmus ohne Pruning, wenn dieser um Pruningfunktionalität erweitert ist? Welche Stopkriterien und welche Parameter für die Kriterien eignen sich besonders?

4. Wie gut sind die Pruningalgorithmen, die in dieser Arbeit implementiert wurden, im Vergleich zum Covering-Algorithmus ohne Pruning, dem Ripper und untereinander?

Am Ende des Kapitels wird anhand der Ergebnisse versucht eine Antwort auf folgende Frage zu geben:

„Ist Pruning überhaupt noch notwendig, wenn die Heuristiken, die zum Lernen verwendet werden, schon sehr gute Ergebnisse erzielen?“

6.3.1 Ergebnisse JRIP vs. Covering

<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>Zeitverhalten in sec</th>
<th>Hypothesengröße in</th>
<th>Genauigkeit in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>JRip</td>
<td>0,20</td>
<td>0,20</td>
<td>0,20</td>
</tr>
<tr>
<td>JRip-P</td>
<td>0,09</td>
<td>0,25</td>
<td>0,27</td>
</tr>
<tr>
<td>JRip-O</td>
<td>0,05</td>
<td>0,05</td>
<td>0,06</td>
</tr>
<tr>
<td>JRip-OP</td>
<td>0,09</td>
<td>0,23</td>
<td>0,25</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
</tbody>
</table>

Tabelle 17: Durchschnittliches Zeitverhalten, durchschnittliche Hypothesengröße und Genauigkeit von JRip und Covering

Für die Vergleiche in diesem Kapitel ist es wichtig zu sagen, dass der Covering-Algorithmus hier gänzlich ohne Pruning getestet wurde. Mit diesem Wissen sind auch die Ergebnisse bei diesen Testläufen zu betrachten. Die Hypothese, die der Algorithmus lernt, wird eine Trainingsmenge garantiert in einem gewissen Maß overfitten. Somit ist das Zeitverhalten, die Größe der Hypothese und die Genauigkeit bei Heuristiken die stark verallgemeinern entsprechend hoch bzw. niedrig. Der Algorithmus versucht jedes positive Beispiel einer Klasse in der Trainingsmenge abzudecken und hat bis auf die doch recht schwachen Pruningkriterien $IncAcc$, $ForwardPruning$ bzw. die verwendete Suchheuristik keine Möglichkeit ein Overfitting an verrauschte Daten zu vermeiden.


In Abbildung 29 ist die Tabelle 17 für die durchschnittlichen Genauigkeiten nochmals visualisiert. In dem Diagramm sind auf der x-Achse die jeweiligen Covering- bzw. JRip-Konfigurationen aufgetragen. Auf der y-Achse erkennt man die durchschnittliche Genauigkeit, die von den Konfigurationen, jeweils auf dem 0%--, 5%- bzw. 10%- Paket, erzielt wurden. Auch hier erkennt man nochmals, dass die Ergebnisse der Konfigurationen JRip-P und JRip-OP gleich sind. Anhand der Abbildung lässt sich feststellen, was ein Rauschen innerhalb der Trainingsdaten bewirkt. Je verrauschter die Daten sind, desto geringer wird auch die Genauigkeit, die ein Lernalgorithmus erzielt. Besonders deutlich wird das, wenn man die Covering Konfigurationen betrachtet, die mit Heuristiken lernen, die zum Overfitting neigen, wie zum Beispiel die Konfigurationen Laplace, Precision oder MEstimate0.5. Die Eigenschaft der Heuristik sehr spezielle Hypothesen zu lernen, liefert auch den

---

12 Diese Tabellen sind auf der beiliegenden CD zu finden. Die Bezeichnungen sind im Anhang A aufgelistet einschließlich der Tabellen, auf die sie sich beziehen.
Grund für das Verhalten, da der Lernalgorithmus mit diesen Heuristiken auch versucht, die verrauschten Daten zu beschreiben. Betrachtet man das Verhalten der Konfigurationen, die mit Heuristiken lernen, die eher allgemeine Hypothesen lernen, wie z.B. WRA, das \textit{m-Estimate} oder \textit{Correlation}, erkennt man das Absinken der Genauigkeiten immer noch, aber nicht mehr in diesem Maße, wie bei den Heuristiken, die stark overfitten.


\subsection{6.3.2 Ergebnisse Prepruning}
Anhand der Tabelle B-1 bis B-4 (siehe Anhang B) erkennt man deutlich den Effekt, den das Prepruning auf das Lernen einer Hypothese hat. Besonders sticht hier ins Auge, dass die Lerngeschwindigkeit auf allen Datenpaketen im Schnitt drastisch erhöht hat und die Hypothesengröße verringert wurde. Betrachtet man jedoch die durchschnittlichen Genauigkeiten auf dem 0%-Paket, erkennt man schnell, dass das Pruning auf nicht verrauschten Daten nur selten etwas oder gar nichts gebracht hat, d.h. dass die durchschnittlichen Genauigkeiten nicht oder nur sehr wenig gestiegen sind. Um dieses Ergebnis zu verdeutlichen dient die Abbildung 30. Hier sind die durchschnittlichen Genauigkeiten der Covering und Prepruning Konfigurationen visualisiert.

Anhand der Abbildung 30 lässt sich schnell sehen, dass das Prepruning bei Konfigurationen, die überspezialisierende Heuristiken verwenden, eher zu einer Verringerung der Genauigkeiten führt. Im
Grunde genommen verringert sich der Grad der Überanpassung der gelernten Hypothesen. Auch bei Heuristiken, die stark verallgemeinern, wie WRA, werden keine Verbesserungen erzielt. Die Konfigurationen, die die Heuristik Klösgen-Maß und das Stopkriterium Significance Testing verwenden, können neben den Konfigurationen Correlation-MDL und MEstimate0.5-Sig95 auf dem 0%-Paket die durchschnittliche Genauigkeit im Vergleich zum Covering erhöhen.

Betrachtet man die Konfigurationen, die als Stopkriterium das MDL-Kriterium verwenden, so erkennt man, dass die Hypothesengröße auf verrauschten Daten erst anwächst und dann wieder abfällt (vgl. Anhang B Tabelle B-1). Das ist auf die Eigenschaft des MDL-Kriteriums zurückzuführen, die versucht Regeln zu finden, die die Datenmenge komprimieren. Je verrauschter die Datenmenge ist, desto weniger Regeln können gefunden werden, die die Datenmenge auch wirklich komprimieren. Also Regeln die viele Beispiele abdecken. Die Regeln, die nur kleine Teile der Trainingsmenge beschreiben, werden abgelehnt und nicht in die Regelmenge aufgenommen. Da das MDL-Kriterium als ClauseStoppingCriterion hier nicht betrachtet worden ist, bedeutet das, dass die gelernten Regeln vom IncAcc-Kriterium im Covering-Algorithmus abgelehnt werden. Es werden also mehr Regeln
gelernt, deren \( fp \)-Anzahl größer als deren \( tp \)-Anzahl ist bzw. es werden mehr leere Regeln, also Regeln deren Körper den Wert true enthält, gelernt.

Ein Blick auf die Tabellen B-3 und B-4 zeigt, wie sich das Significance Testing mit zunehmendem Signifikanzlevel verhält. Im Wesentlichen sinkt die benötigte Zeit zum Lernen und die Größe der Hypothesen nimmt ab. Nichtsdestoweniger werden die durchschnittlichen Genauigkeiten mit dem Significance Testing auf dem 0%-Paket nur für die Heuristiken Klößgen-Maß und m-Estimate mit Parameter 0.5 erhöht (vgl. Abbildung 30).

Im Vergleich zu den beiden vorherigen Stopkriterien ist das Verhalten des CutOff-Kriteriums ähnlich. Ein gravierender Unterschied lässt sich allerdings in den durchschnittlichen Genauigkeiten feststellen. Auf nicht verrauschten Daten, also auf dem 0%-Paket, sind die gelernten Hypothesen maximal genauso gut, wie der Covering-Algorithmus ohne Pruning. Die durchschnittlichen Genauigkeiten pro Heuristik, die mit einem höheren CutOff-Parameter auf dem 0%-Paket gelernt werden, sind zwischen 0% und 5,5% niedriger als die Genauigkeiten des Covering-Algorithmus, wobei die Genauigkeiten mit zunehmenden CutOff-Parameter absinken.


Besonders interessant ist die Tatsache, dass sich die gewählten CutOff-Parameter für WRA und das Klössgen-Maß nicht eigenen, wie man in Abbildung 30 und [TAB-VS] erkennt. Für die Konfigurationen, die mit WRA lernen, werden, egal für welchen CutOff-Parameter, immer die selben Hypothesen gelernt, da das Kriterium bei WRA nur sicherstellt, dass eine Regel mehr \( tp \) als \( fp \) abdeckt, die Bewertung einer Regel im Allgemeinen aber niedriger als 0,3 - bzw. 0,6 oder 0,9 – ausfällt, und das Lernen somit abgebrochen wird, sobald gilt \( tp > fp \). Ähnliches gilt für die Konfigurationen mit dem Klößgen-Maß. Die CutOff-Parameter 0.6 bzw. 0.9 eignen sich hier nicht als Parameter, da die gleichen Hypothesen auf den Datensätzen gelernt werden (vgl. Abbildung 30 bis 32, [TAB-VS]).

Interessant ist auch die Tatsache, dass die Konfigurationen mit Precision-Cut0.3, Laplace-Cut0.3, Mestimate0.5-Cut0.3 und Accuracy-Cut0.3 dieselben Ergebnisse erzielen, wie die Covering Konfigurationen Precision, Laplace und Accuracy. Gleiches gilt auch für das 5%- und 10%-Paket (vgl. Abbildung 30 bis 32). Durch die zusätzliche Bedingung, dass bei jeder Regel \( tp > fp \) gelten muss, werden minimale Heuristikwerte für die drei Heuristiken festgelegt. Seien nun die Gleichungen der Heuristiken wie in Kapitel 3 und die Bedingung \( p = n + 1 \), was gleichbedeutend damit ist, dass eine Regel genau ein positives Beispiel mehr als negative Beispiele abdeckt. Setzt man die Bedingung in die Gleichungen der Heuristiken ein, verändern sich die Heuristiken wie folgt:

\[
\begin{align*}
\text{Precision: } h_{pr}(n, p) &= \frac{n+1}{2n+1} \\
\text{Laplace: } h_{lap}(n, p) &= \frac{n+2}{2n+3} \\
\text{Accuracy: } h_{acc}(n, p) &= \frac{N+1}{N+2}
\end{align*}
\]

\(^{13}\) „maximal“ bedeutet hier, dass der maximale Wert einer Heuristik benutzt wird, z.B. 1 für Laplace, Precision oder Correlation. Für andere Heuristiken müsste der maximale Wert während der Auswertung jedesmal neu berechnet werden.
Abbildung 31: durchschnittliche Genauigkeiten des Preprunings auf dem 5% Paket

Abbildung 31: durchschnittliche Genauigkeiten des Preprunings auf dem 5% Paket
Nimmt man nun an, dass eine Regel unendlich viele negative Beispiele abdeckt, dann sind die minimalen Heuristikwerte für *Precision* und *Laplace* jeweils 0,5. Für *Accuracy* nimmt man an, dass die Trainingsmenge *N+1* positive Beispiele enthält, und erhält als minimalen Wert, wenn eine Regel alle negativen Beispiele abdeckt, auch 0,5. Das CutOff-Kriterium mit dem Parameter 0,3 wird in diesen Konfigurationen also niemals als wahr ausgewertet und es werden somit die gleichen Hypothesen, wie bei den Covering-Konfigurationen ohne Pruning gelernt.

Da das Prepruning auf dem 0%-Paket keine wirkliche Verbesserung gegenüber den Covering-Algorithmus erzielen konnte außer die Verringerung der Laufzeit und der Hypothesengröße, stellt sich nun die Frage, ob das Prepruning auf verrauschten Daten die durchschnittliche Genauigkeit noch erhöhen kann. Um diese Frage zu beantworten dient die Abbildung 31. Die Abbildung visualisiert die durchschnittlichen Genauigkeiten der Prepruning- und Covering-Konfigurationen auf dem 5%-Paket.

Hier erkennt man schon eine Tendenz, die auf dem 10%-Paket erst richtig deutlich wird. Anhand der Abbildung 31 sieht man, dass die durchschnittlichen Genauigkeiten durchaus erhöht werden können, dennoch ist das Prepruning in Verbindung mit Heuristiken, die stark verallgemeinern, wie WRA, nicht in der Lage die Genauigkeiten zu erhöhen. Auch bei den Heuristiken, die stark zum Overfitting neigen kann das Prepruning die Genauigkeiten auf dem 5%-Paket nicht erhöhen. Nichtsdestoweniger zeigt sich, dass, je nachdem welches Stopkriterium verwendet wurde, die Genauigkeiten bei den Konfigurationen *Accuracy*, *Kloesgen*, *Kloesgen0.2*, *Kloesgen0.3*, *MEstimate*, *MEstimate13.97* und *Correlation* erhöht werden können. Betrachtet man in Abbildung 31 die durchschnittliche Genauigkeiten der Konfiguration *MEstimate* und *MEstimate13.97*, sieht man, dass das *m-Estimate* mit einem suboptimalen Parameter bessere Ergebnisse erzielt als das *m-Estimate* mit optimalem Parameter. Durch den suboptimalen Parameter der parametrisierbaren Heuristik overfittet der Lernalgorithmus die Trainingsdaten etwas, was zur Folge hat, dass die Entscheidung des Prunings, das Lernen zu unterbrechen, an anderen Punkten im Suchraum stattfindet. Das Verhalten des Preprunings mit CutOff-Kriterium ist hier ähnlich zu dem Verhalten auf dem 0%-Paket, wie man anhand der Abbildung 31 erkennt.


Abbildung 32: durchschnittliche Genauigkeiten des Preprunings auf dem 10% Paket

Abbildung 32: durchschnittliche Genauigkeiten des Preprunings auf dem 10% Paket
Wie man gesehen hat, ist es durchaus möglich, die Genauigkeiten der Covering-Konfigurationen durch Pruning noch zu erhöhen. Die Frage, die sich hier nun stellt, ist, welcher der Konfigurationen denn nun signifikant besser als die Covering-Konfiguration mit gleicher Heuristik ist. Für die Heuristik Correlation ist dies die Konfiguration Correlation-MDL. Diese Konfiguration ist auf dem 5%-Paket mit 95%-iger Sicherheit und auf dem 10%-Paket mit 99%-iger Sicherheit besser als die Covering-Konfiguration mit Correlation [COV-PRE]. Für das Klösgen-Maß sind das die Konfigurationen in Kombination mit dem Significance Testing und einem Signifikanzniveau von 95% oder 99%, die signifikant bessere Ergebnisse erzielen, als der Covering-Algorithmus mit der gleichen Heuristik. Für das CutOff-Kriterium lässt sich nur für die Heuristik Correlation eine Konfiguration finden, die signifikant besser als Covering mit gleicher Heuristik. Die Konfiguration, die sich hier erkennen lässt, ist Correlation-Cut0.3 [COV-PRE].


Insgesamt lässt sich sagen, dass das Prepruning im Vergleich zum Ripper keine signifikante Verbesserung erzielt, selbst dann nicht wenn Heuristiken zum Lernen benutzt werden, für die bereits optimale Parameter gefunden wurden. Auch bei Heuristiken die stark verallgemeinern oder zum Overfitting neigen, ist der Algorithmus Ripper signifikant besser. Im Vergleich zu den Covering-Konfigurationen wurden durch das Prepruning signifikante Verbesserungen erzielt, vor allem durch die parametrisierbaren Heuristiken und Correlation. Das Prepruning-Stopkriterium, das hier die besten Ergebnisse erzielt, ist das MDL-Kriterium (siehe auch [TAB-VS]). Bei dem Vergleich der Stopkriterien untereinander lässt sich eine Rangfolge erstellen, welches Kriterium am besten geeignet ist. Das MDL-Kriterium, vor allem in Verbindung mit den Heuristiken Klösgen-Maß und Correlation, eignet sich hier am besten [TAB-VS]. Danach folgt das Significance Testing, wobei man feststellt, dass mit steigendem Signifikanzniveau auch bessere Ergebnisse erzielt werden. Vor allem aber in Kombination mit den parametrisierbaren Heuristiken und optimalem Parameter bzw. einem Parameter, der leicht unter dem optimalem Parameter liegt. Am wenigsten geeignet ist hier das CutOff-Kriterium, besonders mit sehr hohem CutOff-Parameter. Die Konfigurationen mit einem CutOff-Parameter von 0.3 sind hier die Konfigurationen, die für das Kriterium die besten Ergebnisse erzielen, insbesondere aber die Konfigurationen MEstimate-Cut0.3 und MEstimate13.97-Cut0.3 [TAB-VS].

6.3.3 Ergebnisse IREP

Aufgrund der stark abfallenden Genauigkeiten wurde für einen weiteren Vergleich auch IREPOpt (vgl. Kapitel 5.7) implementiert, um zu sehen, welche Auswirkung eine abschließende Optimierungsphase der gelernten Hypothese auf die Genauigkeiten hat. Dennoch sind die Ergebnisse
Vergleich von Pruningalgorithmen für Regellerner - Technische Umsetzung im SeCo-Framework


Abbildung 33: durchschnittliche Genauigkeiten von Covering und IREP mit Stopkriterium MaximumErrorRate auf dem 0%-Paket


Abbildung 34: durchschnittliche Genauigkeiten Von Covering und IREP mit Stopkriterium MaximumErrorRate auf dem 5%-Paket

Betrachtet man die Abbildung 35, so erkennt man, dass einige IREP-Konfigurationen bessere Genauigkeiten erzielen konnten als die Covering-Konfigurationen mit gleicher Heuristik. Dennoch sind die Verbesserungen so gering und wenige, dass man diese als statistische Ausreißer ansehen kann. Damit gilt für die IREP-Konfigurationen mit dem Stopkriterium MaximumErrorRate auf dem 10%-Paket dasselbe wie auf dem 0%- und 5%-Paket. D.h., dass selbst auf stark verrauschten Daten durch das Pruning keine Verbesserung erzielt werden konnte.

Eine weitere Frage, die sich stellt, ist, ob es vielleicht ein Stopkriterium gibt, das bessere Ergebnisse erzielt als das *MaximumErrorRate* Stopkriterium. Aus diesem Grund wurden auch die Prepruning Stopkriterien für IREP untersucht. Das zweite Stopkriterium, das für die IREP-Konfigurationen untersucht wurde, ist das MDL-Kriterium. Hier muss allerdings jetzt schon gesagt werden, dass die Ergebnisse nur der Vollständigkeit wegen aufgeführt werden. Wie man an Tabelle B12 (siehe Anhang B) erkennt, werden im Großen und Ganzen nur leere Hypothesen gelernt bzw. im Schnitt ca. eine Regel pro Hypothese. Dies liegt im Wesentlichen daran, dass die Heuristiken Regeln lernen, die nur sehr wenige Beispiele innerhalb der Trainingsmenge abdecken. Deckt eine Regel schon wenige Beispiele der Trainingsmenge ab, so ist es sehr wahrscheinlich, dass auch sehr wenige bis keine Beispiele der Pruningmenge abgedeckt werden. Also wird der Test $L(R)+L(E|R)<L(E)$ nahezu immer als falsch ausgewertet, da die gelernten Regeln die Pruningmenge nicht komprimieren können. Selbst die zusätzlichen Restarts verändern an dem Ergebnis nichts.

In Abbildung 36 sind wiederum die durchschnittlichen Genauigkeiten der Konfigurationen visualisiert. Aber diese Abbildung ist etwas kritischer zu betrachten, da hier aufgrund der sehr schlechten Ergebnisse, die durchschnittliche Genauigkeit über alle 171 Datensätze gezeigt ist, dennoch kann man sich anhand der Tabelle B-12 davon überzeugen, dass keine der IREP-Konfigurationen, die das MDL-Kriterium verwendet haben, bessere Ergebnisse als die Covering-Konfigurationen mit gleicher Heuristik. Die Schwankungen der durchschnittlichen Genauigkeiten liegen, wenn man die einzelnen Datenpakete betrachtet zwischen -7% und -20% im Vergleich zum Covering.

Abbildung 36: durchschnittliche Genauigkeiten von Covering und IREP mit Stopkriterium MDL auf allen Datenpaketen

Abbildung 37: durchschnittliche Genauigkeiten von Covering und IREP mit Stopkriterium Significance Testing auf dem 10%-Paket

Die letzten Konfigurationen für IREP, die hier verglichen werden verwendet als Stopkriterium das absolute CutOff-Kriterium. Wie man an den Tabellen B-6 bis B-8 (Anhang B, blau hinterlegt) erkennt, werden bei den Konfigurationen, die mit dem Klösgen-Maß und WRA lernen, für die CutOff-Parameter 0.3, 0.6 und 0.9 nur leere Hypothesen gelernt. Die einzige Ausnahme sind die IREP-Konfigurationen, die zum Lernen das Klösgen-Maß verwenden, für einen CutOff-Parameter von 0.3.

Auch erkennt man an Abbildung 37, dass zwei IREP-Konfigurationen die Genauigkeiten im Vergleich zum Covering erhöhen konnten. Die Konfigurationen mit den Heuristiken Correlation und Accuracy für einen CutOff-Parameter von 0.3 bzw. bei Accuracy für einen die Parameter 0.3 und 0.6. Auch wenn die Konfigurationen mit dem Klösgen-Maß keine Verbesserung erzielen konnten, sieht man an Abbildung 37 doch, dass es durchaus sinnvoll ist, diese Heuristik mit einem suboptimalen Parameter zu verwenden. Die durchschnittlichen Genauigkeiten steigen mit sinkendem Parameter für das CutOff-Kriterium mit CutOff-Parameter 0.3. Auch sieht man, dass die Genauigkeiten der Konfigurationen, ähnlich denen des Preprunings, mit steigendem CutOff-Parameter sinken.

Abbildung 38: durchschnittliche Genauigkeiten von Covering und IREP mit Stopkriterium absolute CutOff auf dem 10%-Paket


Betrachtet man die WTL Ergebnisse der IREP-Konfigurationen mit MDL-Kriterium für die zwei Operatoren [TAB-VS], zeigt sich, dass die Heuristiken Correlation, Accuracy und WRA für beide Operatoren am geeignetsten sind. Die Konfiguration WRA-FIND-MDL sticht hier besonders hervor, da sie mit 99% Sicherheit besser als die Konfigurationen mit dem delete-last-condition Operator sind.
Welcher Operator aber nun wirklich in Verbindung mit dem MDL-Kriterium am besten geeignet ist, lässt sich auf Grund der generell schlechten Ergebnisse dieser IREP-Konfigurationen nicht feststellen.


Anhand der Tabellen B-6 bis B-8 sieht man, dass die Konfigurationen Correlation-FIND-Cut0.3 und Correlation-DEL-Cut0.3 die höchsten durchschnittlichen Genauigkeiten erzielen. Betrachtet man das Verhalten dieser Konfigurationen für steigende CutOff-Parameter, sieht man, dass die Ergebnisse kontinuierlich schlechter werden. Dieses Verhalten legt nahe, dass der CutOff-Parameter 0.3 für Correlation auch in Verbindung mit IREP ein guter Parameter ist. Aber selbst für diese Konfigurationen gilt, dass die Genauigkeiten im Vergleich zum Covering-Algorithmus erst auf stark verrauschten Daten verbessert werden. Für stark verrauschte Daten ist die Konfiguration Correlation-FIND-Cut0.3 etwas besser als die Konfiguration mit dem delete-last-condition Operator, was durch die WTL-Tabelle [TAB-VS] bestätigt wird.


In groben Zügen lässt sich sagen, dass die IREP-Konfigurationen nur für die parametrisierbaren Heuristiken und Correlation bessere Ergebnisse erzielen als die Prepruning-Konfigurationen, die mit Heuristiken lernen, die zum Overfitting neigen. Besonderes Augenmerk liegt hier auf den IREP-Konfigurationen mit Significance Testing und eingeschränkt auf den Konfigurationen mit CutOff-Kriterium für einen CutOff-Parameter von 0.3 [PRE-IREP].

Die IREP-Konfigurationen, die das Stopkriterium MaximumErrorRate verwenden, schneiden ähnlich zu den Konfigurationen mit dem MDL-Kriterium ab. Diese IREP-Konfigurationen erzielen im Vergleich zu den Prepruning-Konfigurationen nur Verbesserung, sofern die IREP-Konfigurationen mit einer parametrisierbaren Heuristik lernen. Die Verbesserungen werden hier allerdings nur gegen Prepruning-Konfigurationen erzielt, die mit Heuristiken, die zum Overfitting neigen, lernen [PRE-IREP].

Im Folgenden Abschnitt wird versucht für jeden Pruningoperator die besten Konfigurationen zu ermitteln. Die Konfiguration, die sich für den delete-last-condition Operator als am geeignetsten herausstellt ist Correlation-DEL-Sig [TAB-VS]. Diese Konfiguration ist mit 99%-iger Sicherheit
besser als die IREP-Konfigurationen mit einem anderen Stopkriterium [TAB-VS]. Ausnahmen sind hier die Konfigurationen, die mit parametrisierbaren Heuristiken lernen und dasselbe Stopkriterium verwenden, dennoch sieht man hier, dass die Konfiguration Correlation-DEL-Sig mehr Gewinne als Verluste erzielt (gegen die Konfiguration MEstimate-DEL-Max ist die Konfiguration Correlation-DEL-Sig nur mit 95%-iger Sicherheit besser).


Auch die wenigen Konfigurationen, die im Vergleich zu Covering auf verrauschten Daten eine Verbesserung erzielen konnten, sind in ihrer Anzahl und der Verbesserung so gering, dass diese als statistische Ausreißer, nicht nur für das Stopkriterium MaximumErrorRate, sondern für alle Stopkriterien, betrachtet werden können. Betrachtet man dagegen die IREP-Konfigurationen deren durchschnittliche Genauigkeit nahe an denen des Covering mit gleicher Heuristik liegen, so kann man darauf schließen, dass diese Ergebnisse nur erzielt wurden, weil die verwendete Heuristik schon gute Ergebnisse erzielt hat und durch das Prüfen nur eine Verschlechterung erreicht wurde. Anders ausgedrückt, das Pruning konnte an der endgültigen Hypothese nicht viel „kaputt“ machen. Welcher der zwei Pruningoperatoren nun am besten für IREP geeignet ist, lässt sich hier nur bedingt feststellen, da die Ergebnisse unterschiedlich für die Stopkriterien und verwendeten Heuristiken ausfallen. Beide Operatoren sind ungefähr gleich gut geeignet, wobei der delete-last-condition Operator etwas besser abschneidet.

6.4 Ist Pruning überhaupt noch notwendig?
Die Beantwortung der Frage gestaltet sich als nicht einfach, da man nur für eine Pruningmethode eine definitive Aussage treffen kann. Aufgrund der sehr schlechten Ergebnisse der IREP-Konfigurationen lässt sich für diesen Teil die Frage nur bedingt beantworten. Die Testreihen haben hier gezeigt, dass IREP deutlich schlechter abschneidet als der Regellerner Covering mit gleicher Heuristik, was darauf

Für das Prepruning lässt sich eine Tendenz feststellen, die in Verbindung mit dem Grad des Rauschens der Daten steht. Für nicht verrauschte Daten (0% Noise) bringt das Prepruning nichts - weder für übergeneralisierende noch überspezialisierende Heuristiken. Je verrauschter die Daten werden umso mehr eignet sich ein Prunen. Auf den verrauschten Daten (5% bzw. 10% Noise) erkennt man, dass das Prepruning in Kombination mit Heuristiken, die stark overfitten die Ergebnisse noch verbessern kann. Im Schnitt lag die Verbesserung hier bei ca. 2%. Interessant ist hier, dass auch die Ergebnisse, die durch die parametrisierbaren Heuristiken mit optimalem Parameter durch das Prepruning noch verbessert werden konnten. Dies legt die Vermutung nahe, dass ein optimaler Parameter, wie er in [JaFü06] auf nicht verrauschten Daten gefunden wurde, kein Garant dafür ist, dass dieser optimale Parameter auch optimale Ergebnisse auf verrauschten Daten erzielt. Wählt man für die parameterisierbaren Heuristiken einen suboptimalen Parameter, sodass das Lernen mit der Heuristik dazu tendiert etwas speziellere Theorien zu lernen, kann das Prepruning auch die Ergebnisse noch verbessern. Auch die Ergebnisse der einzigen verwendeten, nicht-linearen Heuristik Correlation konnten durch das Prepruning noch verbessert werden. Allerdings ist ein Pruning nicht mehr notwendig, wenn die Heuristik, die verwendet wird stark verallgemeinert.

\footnote{Der Begriff Unstimmigkeit wurde hier gewählt, da es sich um viele Möglichkeiten handeln kann. Dies können zum Beispiel ein Defekt, ein Fehler, ein Bedingung an der falschen Stelle etc. sein.}
Kapitel 7 Schlusswort

7.1 Zusammenfassung
In Kapitel 2 wurden die Separate-and-Conquer Strategie für das Regellernen genauer vorgestellt. Dazu wurde eine kurze Einführung in das Thema des induktiven Lernens gegeben (Kapitel 2.1) und die Merkmale und Eigenschaften der Separate-and-Conquer Algorithmen genauer beleuchtet (Kapitel 2.2). Der Kern, den alle Separate-and-Conquer Algorithmen gemeinsam haben, wurde anhand eines generischen Algorithmus vorgestellt (Kapitel 2.3) und die Grundlagen um die Algorithmen zu vergleichen wurde erklärt (Kapitel 2.4).


Das Pruning wurde eingehend in Kapitel 4 beleuchtet und deren Merkmale sowie Vorteile und Nachteile eingehend beleuchtet. Die ersten Methoden die vorgestellt wurden und wie diese mit verrauschten Daten umgehen, waren das Postpruning (Kapitel 4.1) und das Prepruning (Kapitel 4.2). Die Kapitel 4.3 und 4.4 befassen sich mit der Kombination und der Integration beider Methoden. In Kapitel 4.4 wurden auch die bekanntesten Algorithmen für diese Methode vorgestellt.

Um eine technische Grundlage für die Umsetzung der Pruningalgorithmen zu haben, wurde in Kapitel 5.1 das SeCo-Framework kurz vorgestellt und ein generischer Pruningalgorithmus definiert, der die Prozeduren umfasst, die jedem dieser Algorithmen zugrunde liegen. In einem weiteren Schritt wird die Realisierung der Komponenten (Kapitel 5.2), die für das Pruning notwendig sind, vorgestellt. Der Abschluss des Kapitels widmet sich ausschließlich der Umsetzung der Algorithmen mittels dieser Komponenten (Kapitel 5.3 bis 5.6).

Das sechste Kapitel hält die Ergebnisse des Vergleichs der Pruningalgorithmen fest. Dazu wurden die 57 verwendeten Datensätze des UCI-Repository kurz beschrieben und wie diese mit Rauschen versehen worden sind (Kapitel 6.1). Das Kapitel 6.2 dient der Beschreibung der 268 Konfigurationen, die in dieser Arbeit verglichen wurden. Das Grundgerüst des Vergleichs ist hier durch vier Konfigurationen des RIPPER-Algorithmus und elf Konfigurationen des Covering-Algorithmus gegeben. Die restlichen 253 Konfigurationen sind Erweiterungen des Covering-Algorithmus. In einem ersten Schritt werden die Algorithmen RIPPER und Covering miteinander verglichen (Kapitel 6.3.1), der zweite Schritt erweitert den Vergleich um die Prepruning-Konfigurationen (Kapitel 6.3.2). Der letzte Schritt erweitert den Vergleich um die IREP-Konfigurationen (Kapitel 6.3.3). Das nachfolgende Kapitel ermittelt welche der (Pruning-)Algorithmen die besten Ergebnisse erzielen und vergleicht diese nochmals untereinander um festzustellen, welcher der Algorithmen wirklich der beste ist. Abschließend wird versucht eine Antwort auf die Frage zu geben (Kapitel 6.5):

„Ist Prunen überhaupt noch notwendig, wenn die Heuristiken, die zum Lernen verwendet werden, schon sehr gute Ergebnisse erzielen?“

7.2 Schlussfolgerungen
Das Ziel der Arbeit die verschiedenen Pruningalgorithmen innerhalb eines einheitlichen Regellern-Frameworks, dem SeCo-Framework, zu integrieren ist mit Erfolg erreicht worden. Aufgrund der


Innerhalb des vorliegenden Vergleichs hat sich gezeigt, dass das Prepruning die Pruningstrategie der Wahl ist, da bei 10 von 11 untersuchten Heuristiken eine Steigerung der Genauigkeit möglich ist. Nichtsdestoweniger hat der Vergleich der Algorithmen gezeigt, dass der RIPPER-Algorithmus der momentan beste Pruningalgorithmus für das Regellernen ist, vor allem auf verrauschten Daten.

7.3 Offene Punkte

7.3.1 Heuristiken im Framework


Man erkennt auch, dass es für manche Heuristik-Stopkriterium-Kombination einen optimalen Parameter für das Stopkriterium gibt bzw. dass für manche dieser Kombinationen die gewählten Parameter für das Stopkriterium nicht geeignet waren. Interessant wäre es für diese Kombinationen die optimalen Parameter zu finden.

Interessant wäre es auch für die parametrisierbaren Heuristiken optimale Parameter auf verrauschten Daten zu finden. Die Annahme dahinter ist, dass die Ergebnisse, die die Heuristik mit diesem Parameter erzielt, auch durch das Pruning nicht mehr verbessert werden können. Ein weiterer Aspekt wäre herauszufinden, wie sich die „optimalen“ Parameter auf verrauschten Daten verändern, um mit diesen Informationen vielleicht einen allgemeinen optimalen Parameter für die Heuristiken zu finden.

7.3.2 Pruning im SeCo-Framework

Auch was die Möglichkeiten des Prunings betrifft, die während dieser Arbeit implementiert wurden, ist die Spitze des Eisbergs noch nicht erreicht, d.h. es sind noch nicht alle Pruningalgorithmen, etc. implementiert. Einige Punkte, die noch realisiert werden können, sind:

- Ein anderes auf dem MDL-Prinzip basierendes Stopkriterium. Das hier verwendete Stopkriterium ist an manchen Stellen etwas zu stark gewesen. Interessant wären die Ergebnisse, die man erhält, wenn man ein weniger starkes Kriterium verwenden würde
• Die Erweiterung der vorhandenen Stopkriterien, sodass man diese auch für Regelmengen verwenden kann.
• Aufgrund des Widerspruchs der IREP-Ergebnisse ist es durchaus sinnvoll hier nochmals zu testen, warum dieser Widerspruch zustande kommt.
• Eine Optimierung von Regelmengen, wie sie zum Beispiel im RIPPER-Algorithmus verwendet wird, fehlt noch vollständig.
• Auch sind noch nicht alle Pruningoperatoren vollständig implementiert.

7.3.3 Der vorliegende Vergleich
Selbst für den Vergleich gibt es noch offene Punkte. Einige interessante Aspekte wären z.B.:

• Die Erweiterung des Vergleichs mit den Algorithmen TDP und REP.
• Eine Erweiterung des Vergleichs, sodass auch die Stopkriterien für Regelmengen genauer untersucht werden können.
• Auch wieder eine Erweiterung des Vergleichs, sodass für die Optimierungsphase für Regelmengen hinzugenommen und genauer untersucht werden kann.
• Erweiterung des Vergleichs durch einen Konfiguration, die an den RIPPER-Algorithmus angelehnt ist.
Literaturverzeichnis


## Anhang A – WTL-Verzeichnis

In dieser Tabelle werden kurz die Namen der WTL-Tabellen aufgeführt und wie im Literaturverzeichnis aufgelistet. Jede Tabelle erhält dazu ein bestimmtes Kürzel. Der Ordner, in dem sich die Tabellen auf der CD befinden, heißt „<CD-Laufwerk>:\Tabellen\WTL“.

<table>
<thead>
<tr>
<th>Kürzel</th>
<th>Dateiname</th>
</tr>
</thead>
<tbody>
<tr>
<td>[RIP-COV]</td>
<td>WTL_JRip_Covering.xls</td>
</tr>
<tr>
<td>[RIP-PRE]</td>
<td>WTL_JRip_Prepruning.xls</td>
</tr>
<tr>
<td>[RIP-IREP]</td>
<td>WTL_JRip_IREP.xls</td>
</tr>
<tr>
<td>[COV-PRE]</td>
<td>WTL_Covering_Prepruning.xls</td>
</tr>
<tr>
<td>[COV-IREP]</td>
<td>WTL_Covering_IREP.xls</td>
</tr>
<tr>
<td>[PRE-IREP]</td>
<td>WTL_Prepruning_IREP.xls</td>
</tr>
<tr>
<td>[TAB-VS]</td>
<td>WTL_Algorithmen_gegen_sichselbst.xls</td>
</tr>
</tbody>
</table>
Anhang B – Ergebnistabellen

Anhang B - 1: Ergebnisse Ripper, Covering, Prepruning mit MDL-Kriterium ......................................................... 87
Anhang B - 2: Ergebnisse Covering und Prepruning mit CutOff-Kriterium, CutOff-Parameter 0.3 und 0.6 .... 88
Anhang B - 3: Ergebnisse Covering, Prepruning mit SignificanceTesting, SignifikanzLevel 90% und Prepruning mit CutOff-Kriterium, CutOff-Parameter 0.9 ................................................................. 89
Anhang B - 4: Ergebnisse Covering und Prepruning mit SignificanceTesting, SignifikanzLevel 99% und 95% 90
Anhang B - 5: Ergebnisse Covering und IREP mit MaximumErrorRate ................................................................. 91
Anhang B - 6: Ergebnisse Covering und IREP mit CutOff-Kriterium, CutOff-Parameter 0.6 ......................... 92
Anhang B - 7: Ergebnisse Covering und IREP mit CutOff-Kriterium, CutOff-Parameter 0.6 ......................... 93
Anhang B - 8: Ergebnisse Covering und IREP mit CutOff-Kriterium, CutOff-Parameter 0.9 ......................... 94
Anhang B - 9: Ergebnisse Covering und IREP mit SignificanceTesting, Signifikanzlevel 90% ......................... 95
Anhang B - 10: Ergebnisse Covering und IREP mit SignificanceTesting, Signifikanzlevel 95% ......................... 96
Anhang B - 11: Ergebnisse Covering und IREP mit SignificanceTesting, Signifikanzlevel 99% ......................... 97
Anhang B - 12: Ergebnisse Covering und IREP mit MDL-Kriterium ................................................................. 98
| Konfiguration | Zeitverhalten in sec | Hypothesengröße in $|R|$ | Genauigkeit in % |
|---------------|---------------------|------------------------|-----------------|
|               | 0%-Paket 5%-Paket 10%-Paket | 0%-Paket 5%-Paket 10%-Paket | 0%-Paket 5%-Paket 10%-Paket |
| JRip          | 0,20 0,20 0,20 | 7,30 6,84 6,51 | 81,60 77,34 73,12 |
| JRip-P        | 0,09 0,25 0,27 | 14,70 22,25 22,86 | 81,06 75,12 69,66 |
| JRip-O        | 0,05 0,05 0,06 | 8,68 8,79 9,44 | 80,86 76,03 70,98 |
| JRip-OP       | 0,09 0,23 0,25 | 14,70 22,25 22,86 | 81,06 75,12 69,66 |
| Laplace       | 1,45 3,47 5,54 | 41,07 60,72 78,51 | 79,43 72,67 66,57 |
| Precision     | 1,60 3,92 6,06 | 45,82 70,95 88,60 | 79,13 72,57 66,36 |
| Accuracy      | 1,47 2,42 3,46 | 35,37 42,44 54,79 | 78,78 73,36 67,50 |
| WRAcc         | 0,57 0,87 1,02 | 4,25 4,46 4,39 | 79,16 74,93 70,27 |
| Kloesgen      | 0,98 2,21 1,81 | 15,32 30,60 31,58 | 81,47 75,39 69,62 |
| Kloesgen0.3   | 1,19 2,42 3,52 | 29,07 40,07 49,04 | 80,70 74,82 68,88 |
| Kloesgen0.2   | 1,34 2,50 3,78 | 35,44 45,19 59,04 | 80,17 74,06 67,73 |
| MEstimate     | 0,88 1,29 1,60 | 13,25 16,46 18,53 | 82,08 75,41 70,85 |
| MEstimate 0.5 | 1,41 3,60 5,77 | 42,30 64,33 83,09 | 79,20 72,95 66,84 |
| MEstimate 13.97 | 0,80 1,31 1,58 | 15,30 20,54 22,75 | 81,91 75,88 70,36 |
| Correlation   | 0,89 1,28 1,60 | 14,42 18,72 22,07 | 80,46 75,52 69,97 |
| Laplace-MDL   | 0,18 0,20 0,19 | 13,11 14,19 14,07 | 76,63 71,85 68,10 |
| Precision-MDL | 0,24 0,26 0,25 | 17,02 18,63 18,39 | 76,17 71,40 67,74 |
| Accuracy-MDL  | 0,23 0,26 0,28 | 7,35 7,88 7,75 | 77,27 73,45 69,66 |
| WRAcc-MDL     | 0,48 0,71 0,79 | 4,16 4,19 4,09 | 78,97 74,77 70,19 |
| Kloesgen-MDL  | 0,35 0,42 0,41 | 9,00 9,47 8,74 | 80,54 75,77 71,23 |
| Kloesgen0.3-MDL | 0,26 0,38 0,37 | 9,96 11,28 11,23 | 79,60 75,11 70,70 |
| Kloesgen0.2-MDL | 0,24 0,34 0,32 | 12,56 12,79 12,28 | 78,25 74,28 69,59 |
| MEstimate-MDL | 0,36 0,45 0,40 | 7,95 8,30 7,46 | 80,04 74,74 70,60 |
| MEstimate 0.5-MDL | 0,22 0,24 0,23 | 15,18 16,14 16,32 | 76,65 71,99 68,31 |
| MEstimate 13.97-MDL | 0,28 0,34 0,35 | 8,04 8,18 7,96 | 79,40 74,78 70,24 |
| Correlation-MDL | 0,43 0,52 0,54 | 8,37 8,11 8,81 | 80,49 76,10 71,71 |

Anhang B - 1: Ergebnisse Ripper, Covering, Pruning mit MDL-Kriterium
<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>Zeitverhalten in sec</th>
<th>Hypothesengröße in</th>
<th>Genauigkeit in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRacc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-CUT0.3</td>
<td>1,35</td>
<td>3,29</td>
<td>5,37</td>
</tr>
<tr>
<td>Precision-CUT0.3</td>
<td>1,49</td>
<td>3,76</td>
<td>5,85</td>
</tr>
<tr>
<td>Accuracy-CUT0.3</td>
<td>1,41</td>
<td>2,32</td>
<td>3,31</td>
</tr>
<tr>
<td>WRAcc-CUT0.3</td>
<td>0,21</td>
<td>0,39</td>
<td>0,46</td>
</tr>
<tr>
<td>Kloesgen-CUT0.3</td>
<td>0,26</td>
<td>0,38</td>
<td>0,45</td>
</tr>
<tr>
<td>Kloesgen0.3-CUT0.3</td>
<td>0,31</td>
<td>0,46</td>
<td>0,51</td>
</tr>
<tr>
<td>Kloesgen0.2-CUT0.3</td>
<td>0,39</td>
<td>0,52</td>
<td>0,60</td>
</tr>
<tr>
<td>MEstimate-CUT0.3</td>
<td>0,57</td>
<td>0,87</td>
<td>1,03</td>
</tr>
<tr>
<td>MEstimate 0.5-CUT0.3</td>
<td>1,41</td>
<td>3,51</td>
<td>5,66</td>
</tr>
<tr>
<td>MEstimate 13.97-CUT0.3</td>
<td>0,60</td>
<td>0,95</td>
<td>1,16</td>
</tr>
<tr>
<td>Correlation-CUT0.3</td>
<td>0,64</td>
<td>0,85</td>
<td>0,92</td>
</tr>
<tr>
<td>Laplace-CUT0.6</td>
<td>0,33</td>
<td>0,78</td>
<td>0,94</td>
</tr>
<tr>
<td>Precision-CUT0.6</td>
<td>0,39</td>
<td>0,89</td>
<td>1,11</td>
</tr>
<tr>
<td>Accuracy-CUT0.6</td>
<td>1,40</td>
<td>2,32</td>
<td>3,30</td>
</tr>
<tr>
<td>WRAcc-CUT0.6</td>
<td>0,20</td>
<td>0,39</td>
<td>0,46</td>
</tr>
<tr>
<td>Kloesgen-CUT0.6</td>
<td>0,24</td>
<td>0,31</td>
<td>0,35</td>
</tr>
<tr>
<td>Kloesgen0.3-CUT0.6</td>
<td>0,26</td>
<td>0,33</td>
<td>0,33</td>
</tr>
<tr>
<td>Kloesgen0.2-CUT0.6</td>
<td>0,25</td>
<td>0,31</td>
<td>0,38</td>
</tr>
<tr>
<td>MEstimate-CUT0.6</td>
<td>0,26</td>
<td>0,47</td>
<td>0,54</td>
</tr>
<tr>
<td>MEstimate 0.5-CUT0.6</td>
<td>0,33</td>
<td>0,83</td>
<td>1,02</td>
</tr>
<tr>
<td>MEstimate 13.97-CUT0.6</td>
<td>0,30</td>
<td>0,52</td>
<td>0,59</td>
</tr>
<tr>
<td>Correlation-CUT0.6</td>
<td>0,33</td>
<td>0,49</td>
<td>0,49</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitverhalten in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-CUT0.9</td>
<td>0,18</td>
<td>0,37</td>
<td>0,26</td>
</tr>
<tr>
<td>Precision-CUT0.9</td>
<td>0,21</td>
<td>0,41</td>
<td>0,31</td>
</tr>
<tr>
<td>Accuracy-CUT0.9</td>
<td>0,72</td>
<td>1,27</td>
<td>0,66</td>
</tr>
<tr>
<td>WRAcc-CUT0.9</td>
<td>0,20</td>
<td>0,39</td>
<td>0,46</td>
</tr>
<tr>
<td>Kloesgen-CUT0.9</td>
<td>0,24</td>
<td>0,31</td>
<td>0,34</td>
</tr>
<tr>
<td>Kloesgen0.3-CUT0.9</td>
<td>0,26</td>
<td>0,33</td>
<td>0,33</td>
</tr>
<tr>
<td>Kloesgen0.2-CUT0.9</td>
<td>0,25</td>
<td>0,31</td>
<td>0,38</td>
</tr>
<tr>
<td>MEstimate-CUT0.9</td>
<td>0,20</td>
<td>0,30</td>
<td>0,29</td>
</tr>
<tr>
<td>MEstimate 0.5-CUT0.9</td>
<td>0,20</td>
<td>0,37</td>
<td>0,28</td>
</tr>
<tr>
<td>MEstimate 13.97-CUT0.9</td>
<td>0,19</td>
<td>0,31</td>
<td>0,29</td>
</tr>
<tr>
<td>Correlation-CUT0.9</td>
<td>0,18</td>
<td>0,32</td>
<td>0,32</td>
</tr>
<tr>
<td>Laplace-Sig09</td>
<td>0,90</td>
<td>2,47</td>
<td>1,95</td>
</tr>
<tr>
<td>Precision-Sig09</td>
<td>1,03</td>
<td>2,89</td>
<td>2,21</td>
</tr>
<tr>
<td>Accuracy-Sig09</td>
<td>1,04</td>
<td>1,63</td>
<td>2,15</td>
</tr>
<tr>
<td>WRAcc-Sig09</td>
<td>0,54</td>
<td>0,81</td>
<td>0,93</td>
</tr>
<tr>
<td>Kloesgen-Sig09</td>
<td>0,84</td>
<td>2,09</td>
<td>1,57</td>
</tr>
<tr>
<td>Kloesgen0.3-Sig09</td>
<td>1,05</td>
<td>2,15</td>
<td>3,05</td>
</tr>
<tr>
<td>Kloesgen0.2-Sig09</td>
<td>1,06</td>
<td>2,17</td>
<td>2,67</td>
</tr>
<tr>
<td>MEstimate-Sig09</td>
<td>0,80</td>
<td>1,20</td>
<td>1,48</td>
</tr>
<tr>
<td>MEstimate 0.5-Sig09</td>
<td>0,99</td>
<td>2,64</td>
<td>2,12</td>
</tr>
<tr>
<td>MEstimate 13.97-Sig09</td>
<td>0,79</td>
<td>1,31</td>
<td>1,55</td>
</tr>
<tr>
<td>Correlation-Sig09</td>
<td>0,82</td>
<td>1,19</td>
<td>1,46</td>
</tr>
</tbody>
</table>

Anhang B - 3: Ergebnisse Covering, Prepruning mit Significance Testing, Signifikanz Level 90% und Prepruning mit CutOff-Kriterium, CutOff-Parameter 0.9
<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>Zeitverhalten in sec</th>
<th>Hypothesengröße in</th>
<th>Genauigkeit in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,07</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,85</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-Sig95</td>
<td>0,83</td>
<td>1,40</td>
<td>1,82</td>
</tr>
<tr>
<td>Precision-Sig95</td>
<td>0,92</td>
<td>1,67</td>
<td>1,98</td>
</tr>
<tr>
<td>Accuracy-Sig95</td>
<td>0,92</td>
<td>1,41</td>
<td>0,95</td>
</tr>
<tr>
<td>WRAcc-Sig95</td>
<td>0,53</td>
<td>0,79</td>
<td>0,93</td>
</tr>
<tr>
<td>Kloesgen-Sig95</td>
<td>0,79</td>
<td>2,01</td>
<td>1,36</td>
</tr>
<tr>
<td>Kloesgen0.3-Sig95</td>
<td>0,95</td>
<td>2,01</td>
<td>1,24</td>
</tr>
<tr>
<td>Kloesgen0.2-Sig95</td>
<td>1,00</td>
<td>1,99</td>
<td>1,28</td>
</tr>
<tr>
<td>MEstimate-Sig95</td>
<td>0,80</td>
<td>1,18</td>
<td>1,44</td>
</tr>
<tr>
<td>MEstimate 0.5-Sig95</td>
<td>0,88</td>
<td>1,57</td>
<td>2,03</td>
</tr>
<tr>
<td>MEstimate 13.97-Sig95</td>
<td>0,77</td>
<td>1,28</td>
<td>1,47</td>
</tr>
<tr>
<td>Correlation-Sig95</td>
<td>0,78</td>
<td>1,11</td>
<td>1,44</td>
</tr>
<tr>
<td>Laplace-Sig99</td>
<td>0,63</td>
<td>1,12</td>
<td>1,42</td>
</tr>
<tr>
<td>Precision-Sig99</td>
<td>0,75</td>
<td>1,36</td>
<td>1,83</td>
</tr>
<tr>
<td>Accuracy-Sig99</td>
<td>0,61</td>
<td>0,69</td>
<td>0,77</td>
</tr>
<tr>
<td>WRAcc-Sig99</td>
<td>0,53</td>
<td>0,80</td>
<td>0,93</td>
</tr>
<tr>
<td>Kloesgen-Sig99</td>
<td>0,72</td>
<td>1,05</td>
<td>1,17</td>
</tr>
<tr>
<td>Kloesgen0.3-Sig99</td>
<td>0,80</td>
<td>0,92</td>
<td>0,98</td>
</tr>
<tr>
<td>Kloesgen0.2-Sig99</td>
<td>0,81</td>
<td>0,85</td>
<td>1,02</td>
</tr>
<tr>
<td>MEstimate-Sig99</td>
<td>0,79</td>
<td>1,14</td>
<td>1,39</td>
</tr>
<tr>
<td>MEstimate 0.5-Sig99</td>
<td>0,72</td>
<td>1,34</td>
<td>1,70</td>
</tr>
<tr>
<td>MEstimate 13.97-Sig99</td>
<td>0,74</td>
<td>1,18</td>
<td>1,38</td>
</tr>
<tr>
<td>Correlation-Sig99</td>
<td>0,74</td>
<td>1,00</td>
<td>1,15</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitspanne in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-FIND-Max</td>
<td>0,34</td>
<td>0,43</td>
<td>0,48</td>
</tr>
<tr>
<td>Precision-FIND-Max</td>
<td>0,53</td>
<td>0,71</td>
<td>0,78</td>
</tr>
<tr>
<td>Accuracy-FIND-Max</td>
<td>0,46</td>
<td>0,51</td>
<td>0,54</td>
</tr>
<tr>
<td>WRAcc-FIND-Max</td>
<td>0,46</td>
<td>0,67</td>
<td>0,75</td>
</tr>
<tr>
<td>Kloesgen-FIND-Max</td>
<td>0,48</td>
<td>0,66</td>
<td>0,82</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Max</td>
<td>0,51</td>
<td>0,61</td>
<td>0,81</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Max</td>
<td>0,53</td>
<td>0,64</td>
<td>0,85</td>
</tr>
<tr>
<td>MEstimate-FIND-Max</td>
<td>0,49</td>
<td>0,72</td>
<td>0,79</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-Max</td>
<td>0,45</td>
<td>0,62</td>
<td>0,67</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-Max</td>
<td>0,49</td>
<td>0,68</td>
<td>0,80</td>
</tr>
<tr>
<td>Correlation-FIND-Max</td>
<td>0,52</td>
<td>0,76</td>
<td>0,79</td>
</tr>
<tr>
<td>Laplace-DEL-Max</td>
<td>0,35</td>
<td>0,45</td>
<td>0,53</td>
</tr>
<tr>
<td>Precision-DEL-Max</td>
<td>0,37</td>
<td>0,49</td>
<td>0,52</td>
</tr>
<tr>
<td>Accuracy-DEL-Max</td>
<td>0,48</td>
<td>0,54</td>
<td>0,59</td>
</tr>
<tr>
<td>WRAcc-DEL-Max</td>
<td>0,49</td>
<td>0,67</td>
<td>0,80</td>
</tr>
<tr>
<td>Kloesgen-DEL-Max</td>
<td>0,56</td>
<td>0,68</td>
<td>0,86</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Max</td>
<td>0,50</td>
<td>0,68</td>
<td>0,78</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Max</td>
<td>0,46</td>
<td>0,60</td>
<td>0,68</td>
</tr>
<tr>
<td>MEstimate-DEL-Max</td>
<td>0,50</td>
<td>0,70</td>
<td>0,88</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Max</td>
<td>0,38</td>
<td>0,47</td>
<td>0,55</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Max</td>
<td>0,52</td>
<td>0,72</td>
<td>0,76</td>
</tr>
<tr>
<td>Correlation-DEL-Max</td>
<td>0,54</td>
<td>0,74</td>
<td>0,81</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitverhalten in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket 5%-Paket 10%-Paket</td>
<td>0%-Paket 5%-Paket 10%-Paket</td>
<td>0%-Paket 5%-Paket 10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45 3,47 5,54</td>
<td>41,07 60,72 78,51</td>
<td>79,43 72,67 66,57</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60 3,92 6,06</td>
<td>45,82 70,95 88,60</td>
<td>79,13 72,57 66,36</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47 2,42 3,46</td>
<td>35,37 42,44 54,79</td>
<td>78,78 73,36 67,50</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57 0,87 1,02</td>
<td>4,25 4,46 4,39</td>
<td>79,16 74,93 70,27</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98 2,21 1,81</td>
<td>15,32 30,60 31,58</td>
<td>81,47 75,39 69,62</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19 2,42 3,52</td>
<td>29,07 40,07 49,04</td>
<td>80,70 74,82 68,88</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34 2,50 3,78</td>
<td>35,44 45,19 59,04</td>
<td>80,17 74,06 67,73</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88 1,29 1,60</td>
<td>13,25 16,46 18,53</td>
<td>82,08 75,41 70,85</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41 3,60 5,77</td>
<td>42,30 64,33 83,09</td>
<td>79,20 72,95 66,84</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80 1,31 1,58</td>
<td>15,30 20,54 22,75</td>
<td>81,91 75,88 70,36</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89 1,28 1,60</td>
<td>14,42 18,72 22,07</td>
<td>80,46 75,52 69,97</td>
</tr>
<tr>
<td>Laplace-FIND-Cut0.3</td>
<td>0,29 0,46 0,45</td>
<td>4,25 4,95 4,84</td>
<td>73,41 69,17 64,76</td>
</tr>
<tr>
<td>Precision-FIND-Cut0.3</td>
<td>0,26 0,36 0,39</td>
<td>3,84 3,56 3,67</td>
<td>70,11 64,63 60,15</td>
</tr>
<tr>
<td>Accuracy-FIND-Cut0.3</td>
<td>0,45 0,60 0,65</td>
<td>3,61 3,25 3,12</td>
<td>76,03 71,77 68,17</td>
</tr>
<tr>
<td>WRAcc-FIND-Cut0.3</td>
<td>0,44 0,64 0,69</td>
<td>0,00 0,00 0,00</td>
<td>54,29 52,88 51,49</td>
</tr>
<tr>
<td>Kloesgen-FIND-Cut0.3</td>
<td>0,29 0,43 0,43</td>
<td>0,47 0,33 0,23</td>
<td>62,49 58,95 54,95</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Cut0.3</td>
<td>0,27 0,43 0,43</td>
<td>1,72 1,54 1,00</td>
<td>70,12 65,45 59,99</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Cut0.3</td>
<td>0,29 0,44 0,47</td>
<td>3,39 3,02 2,96</td>
<td>73,21 68,44 62,86</td>
</tr>
<tr>
<td>MEstimate-FIND-Cut0.3</td>
<td>0,56 0,83 0,91</td>
<td>3,72 3,81 3,65</td>
<td>76,99 72,35 69,32</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-Cut0.3</td>
<td>0,28 0,40 0,42</td>
<td>4,26 4,49 4,11</td>
<td>71,64 66,99 62,09</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-Cut0.3</td>
<td>0,56 0,79 0,90</td>
<td>4,23 4,18 4,18</td>
<td>77,79 73,36 69,79</td>
</tr>
<tr>
<td>Correlation-FIND-Cut0.3</td>
<td>0,63 0,77 0,82</td>
<td>4,04 3,51 3,44</td>
<td>78,83 74,13 70,48</td>
</tr>
<tr>
<td>Laplace-DEL-Cut0.3</td>
<td>0,29 0,43 0,43</td>
<td>3,82 3,96 4,30</td>
<td>72,63 68,42 64,49</td>
</tr>
<tr>
<td>Precision-DEL-Cut0.3</td>
<td>0,26 0,36 0,40</td>
<td>3,34 3,98 3,18</td>
<td>69,38 64,07 59,88</td>
</tr>
<tr>
<td>Accuracy-DEL-Cut0.3</td>
<td>0,46 0,60 0,71</td>
<td>3,59 3,63 3,60</td>
<td>76,03 72,23 68,18</td>
</tr>
<tr>
<td>WRAcc-DEL-Cut0.3</td>
<td>0,44 0,64 0,69</td>
<td>0,00 0,00 0,00</td>
<td>53,90 52,88 51,49</td>
</tr>
<tr>
<td>Kloesgen-DEL-Cut0.3</td>
<td>0,30 0,44 0,44</td>
<td>0,50 0,32 0,21</td>
<td>62,38 58,91 55,07</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Cut0.3</td>
<td>0,29 0,47 0,46</td>
<td>1,68 1,42 1,11</td>
<td>69,07 65,10 59,98</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Cut0.3</td>
<td>0,31 0,44 0,50</td>
<td>2,95 2,37 2,32</td>
<td>72,10 67,27 62,32</td>
</tr>
<tr>
<td>MEstimate-DEL-Cut0.3</td>
<td>0,62 0,84 0,89</td>
<td>4,04 3,70 3,56</td>
<td>76,70 72,65 68,95</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Cut0.3</td>
<td>0,29 0,39 0,40</td>
<td>3,84 3,75 3,61</td>
<td>71,10 66,57 61,92</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Cut0.3</td>
<td>0,60 0,82 0,96</td>
<td>4,50 4,18 4,28</td>
<td>77,32 73,35 69,33</td>
</tr>
<tr>
<td>Correlation-DEL-Cut0.3</td>
<td>0,69 0,78 0,92</td>
<td>4,14 3,51 3,44</td>
<td>78,76 74,65 70,45</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitverhalten in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1.45</td>
<td>3.47</td>
<td>5.54</td>
</tr>
<tr>
<td>Precision</td>
<td>1.60</td>
<td>3.92</td>
<td>6.06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1.47</td>
<td>2.42</td>
<td>3.46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0.57</td>
<td>0.87</td>
<td>1.02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0.98</td>
<td>2.21</td>
<td>1.81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1.19</td>
<td>2.42</td>
<td>3.52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1.34</td>
<td>2.50</td>
<td>3.78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0.88</td>
<td>1.29</td>
<td>1.60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1.41</td>
<td>3.60</td>
<td>5.77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0.80</td>
<td>1.31</td>
<td>1.58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0.89</td>
<td>1.28</td>
<td>1.60</td>
</tr>
<tr>
<td>Laplace-FIND-Cut0.6</td>
<td>0.27</td>
<td>0.44</td>
<td>0.42</td>
</tr>
<tr>
<td>Precision-FIND-Cut0.6</td>
<td>0.25</td>
<td>0.35</td>
<td>0.39</td>
</tr>
<tr>
<td>Accuracy-FIND-Cut0.6</td>
<td>0.45</td>
<td>0.60</td>
<td>0.66</td>
</tr>
<tr>
<td>WRAcc-FIND-Cut0.6</td>
<td>0.44</td>
<td>0.64</td>
<td>0.69</td>
</tr>
<tr>
<td>Kloesgen-FIND-Cut0.6</td>
<td>0.26</td>
<td>0.39</td>
<td>0.42</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Cut0.6</td>
<td>0.22</td>
<td>0.33</td>
<td>0.39</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Cut0.6</td>
<td>0.18</td>
<td>0.28</td>
<td>0.31</td>
</tr>
<tr>
<td>MEstimate-FIND-Cut0.6</td>
<td>0.38</td>
<td>0.52</td>
<td>0.57</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-Cut0.6</td>
<td>0.26</td>
<td>0.39</td>
<td>0.42</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-Cut0.6</td>
<td>0.36</td>
<td>0.49</td>
<td>0.59</td>
</tr>
<tr>
<td>Correlation-FIND-Cut0.6</td>
<td>0.47</td>
<td>0.62</td>
<td>0.66</td>
</tr>
<tr>
<td>Laplace-DEL-Cut0.6</td>
<td>0.28</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td>Precision-DEL-Cut0.6</td>
<td>0.26</td>
<td>0.35</td>
<td>0.38</td>
</tr>
<tr>
<td>Accuracy-DEL-Cut0.6</td>
<td>0.45</td>
<td>0.61</td>
<td>0.71</td>
</tr>
<tr>
<td>WRAcc-DEL-Cut0.6</td>
<td>0.44</td>
<td>0.64</td>
<td>0.69</td>
</tr>
<tr>
<td>Kloesgen-DEL-Cut0.6</td>
<td>0.23</td>
<td>0.39</td>
<td>0.44</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Cut0.6</td>
<td>0.23</td>
<td>0.35</td>
<td>0.39</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Cut0.6</td>
<td>0.20</td>
<td>0.31</td>
<td>0.33</td>
</tr>
<tr>
<td>MEstimate-DEL-Cut0.6</td>
<td>0.39</td>
<td>0.54</td>
<td>0.58</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Cut0.6</td>
<td>0.27</td>
<td>0.36</td>
<td>0.39</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Cut0.6</td>
<td>0.39</td>
<td>0.53</td>
<td>0.62</td>
</tr>
<tr>
<td>Correlation-DEL-Cut0.6</td>
<td>0.48</td>
<td>0.63</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Anhang B - 7: Ergebnisse Covering und IREP mit CutOff-Kriterium, CutOff-Parameter 0.6
<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>Zeitverhalten in sec</th>
<th>Hypothesengröße in</th>
<th>Genauigkeit in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-FIND-Cut0.9</td>
<td>0,18</td>
<td>0,24</td>
<td>0,24</td>
</tr>
<tr>
<td>Precision-FIND-Cut0.9</td>
<td>0,21</td>
<td>0,29</td>
<td>0,28</td>
</tr>
<tr>
<td>Accuracy-FIND-Cut0.9</td>
<td>0,33</td>
<td>0,48</td>
<td>0,52</td>
</tr>
<tr>
<td>WRAcc-FIND-Cut0.9</td>
<td>0,44</td>
<td>0,64</td>
<td>0,69</td>
</tr>
<tr>
<td>Kloesgen-Cut0.9</td>
<td>0,27</td>
<td>0,39</td>
<td>0,42</td>
</tr>
<tr>
<td>Kloesgen0.3-Cut0.9</td>
<td>0,22</td>
<td>0,34</td>
<td>0,39</td>
</tr>
<tr>
<td>Kloesgen0.2-Cut0.9</td>
<td>0,18</td>
<td>0,29</td>
<td>0,31</td>
</tr>
<tr>
<td>MEstimate-Cut0.9</td>
<td>0,30</td>
<td>0,41</td>
<td>0,46</td>
</tr>
<tr>
<td>MEstimate 0.5-CUT-Cut0.9</td>
<td>0,18</td>
<td>0,27</td>
<td>0,25</td>
</tr>
<tr>
<td>MEstimate 13.97-CUT-Cut0.9</td>
<td>0,28</td>
<td>0,41</td>
<td>0,43</td>
</tr>
<tr>
<td>Correlation-Cut0.9</td>
<td>0,37</td>
<td>0,51</td>
<td>0,49</td>
</tr>
<tr>
<td>Laplace-DEL-Cut0.9</td>
<td>0,18</td>
<td>0,24</td>
<td>0,23</td>
</tr>
<tr>
<td>Precision-DEL-Cut0.9</td>
<td>0,23</td>
<td>0,27</td>
<td>0,28</td>
</tr>
<tr>
<td>Accuracy-DEL-Cut0.9</td>
<td>0,33</td>
<td>0,48</td>
<td>0,53</td>
</tr>
<tr>
<td>WRAcc-DEL-Cut0.9</td>
<td>0,44</td>
<td>0,39</td>
<td>0,69</td>
</tr>
<tr>
<td>Kloesgen-DEL-Cut0.9</td>
<td>0,27</td>
<td>0,39</td>
<td>0,38</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Cut0.9</td>
<td>0,23</td>
<td>0,35</td>
<td>0,32</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Cut0.9</td>
<td>0,20</td>
<td>0,31</td>
<td>0,45</td>
</tr>
<tr>
<td>MEstimate-DEL-Cut0.9</td>
<td>0,30</td>
<td>0,41</td>
<td>0,45</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Cut0.9</td>
<td>0,20</td>
<td>0,25</td>
<td>0,24</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Cut0.9</td>
<td>0,28</td>
<td>0,40</td>
<td>0,42</td>
</tr>
<tr>
<td>Correlation-DEL-Cut0.9</td>
<td>0,38</td>
<td>0,52</td>
<td>0,48</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitverhalten in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.5</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEestimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEestimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEestimate 13.97</td>
<td>1,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-FIND-Sig90</td>
<td>0,29</td>
<td>0,46</td>
<td>0,45</td>
</tr>
<tr>
<td>Precision-FIND-Sig90</td>
<td>0,26</td>
<td>0,36</td>
<td>0,39</td>
</tr>
<tr>
<td>Accuracy-FIND-Sig90</td>
<td>0,45</td>
<td>0,60</td>
<td>0,65</td>
</tr>
<tr>
<td>WRAcc-FIND-Sig90</td>
<td>0,69</td>
<td>1,24</td>
<td>1,39</td>
</tr>
<tr>
<td>Kloesgen-FIND-Sig90</td>
<td>0,51</td>
<td>0,71</td>
<td>0,77</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Sig90</td>
<td>0,40</td>
<td>0,59</td>
<td>0,63</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Sig90</td>
<td>0,33</td>
<td>0,50</td>
<td>0,58</td>
</tr>
<tr>
<td>MEestimate-FIND-Sig90</td>
<td>0,70</td>
<td>1,01</td>
<td>1,12</td>
</tr>
<tr>
<td>MEestimate 0.5-FIND-Sig90</td>
<td>0,28</td>
<td>0,40</td>
<td>0,42</td>
</tr>
<tr>
<td>MEestimate 13.97-FIND-Sig90</td>
<td>0,59</td>
<td>0,85</td>
<td>1,00</td>
</tr>
<tr>
<td>Correlation-FIND-Sig90</td>
<td>0,63</td>
<td>0,81</td>
<td>0,87</td>
</tr>
<tr>
<td>Laplace-DEL-Sig90</td>
<td>0,28</td>
<td>0,43</td>
<td>0,44</td>
</tr>
<tr>
<td>Precision-DEL-Sig90</td>
<td>0,27</td>
<td>0,36</td>
<td>0,40</td>
</tr>
<tr>
<td>Accuracy-DEL-Sig90</td>
<td>0,45</td>
<td>0,61</td>
<td>0,71</td>
</tr>
<tr>
<td>WRAcc-DEL-Sig90</td>
<td>0,76</td>
<td>1,33</td>
<td>1,49</td>
</tr>
<tr>
<td>Kloesgen-DEL-Sig90</td>
<td>0,57</td>
<td>0,80</td>
<td>0,82</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Sig90</td>
<td>0,44</td>
<td>0,61</td>
<td>0,65</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Sig90</td>
<td>0,36</td>
<td>0,53</td>
<td>0,57</td>
</tr>
<tr>
<td>MEestimate-DEL-Sig90</td>
<td>0,77</td>
<td>1,06</td>
<td>1,15</td>
</tr>
<tr>
<td>MEestimate 0.5-DEL-Sig90</td>
<td>0,28</td>
<td>0,39</td>
<td>0,40</td>
</tr>
<tr>
<td>MEestimate 13.97-DEL-Sig90</td>
<td>0,67</td>
<td>0,91</td>
<td>1,01</td>
</tr>
<tr>
<td>Correlation-DEL-Sig90</td>
<td>0,71</td>
<td>0,86</td>
<td>1,01</td>
</tr>
</tbody>
</table>

Anhang B - 9: Ergebnisse Covering und IREP mit Significance Testing, Signifikanzlevel 90%
<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>Zeitverhalten in sec</th>
<th>Hypothesengröße in</th>
<th>Genauigkeit in %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-FIND-Sig95</td>
<td>0,29</td>
<td>0,46</td>
<td>0,44</td>
</tr>
<tr>
<td>Precision-FIND-Sig95</td>
<td>0,25</td>
<td>0,36</td>
<td>0,39</td>
</tr>
<tr>
<td>Accuracy-FIND-Sig95</td>
<td>0,45</td>
<td>0,60</td>
<td>0,65</td>
</tr>
<tr>
<td>WRAcc-FIND-Sig95</td>
<td>0,69</td>
<td>1,24</td>
<td>1,39</td>
</tr>
<tr>
<td>Kloesgen-FIND-Sig95</td>
<td>0,51</td>
<td>0,72</td>
<td>0,77</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Sig95</td>
<td>0,40</td>
<td>0,58</td>
<td>0,63</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Sig95</td>
<td>0,34</td>
<td>0,50</td>
<td>0,58</td>
</tr>
<tr>
<td>MEstimate-FIND-Sig95</td>
<td>0,70</td>
<td>1,01</td>
<td>1,12</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-Sig95</td>
<td>0,27</td>
<td>0,40</td>
<td>0,42</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-Sig95</td>
<td>0,59</td>
<td>0,85</td>
<td>1,01</td>
</tr>
<tr>
<td>Correlation-FIND-Sig95</td>
<td>0,63</td>
<td>0,80</td>
<td>0,87</td>
</tr>
<tr>
<td>Laplace-DEL-Sig95</td>
<td>0,28</td>
<td>0,44</td>
<td>0,44</td>
</tr>
<tr>
<td>Precision-DEL-Sig95</td>
<td>0,26</td>
<td>0,36</td>
<td>0,40</td>
</tr>
<tr>
<td>Accuracy-DEL-Sig95</td>
<td>0,45</td>
<td>0,61</td>
<td>0,71</td>
</tr>
<tr>
<td>WRAcc-DEL-Sig95</td>
<td>0,75</td>
<td>1,34</td>
<td>1,49</td>
</tr>
<tr>
<td>Kloesgen-DEL-Sig95</td>
<td>0,57</td>
<td>0,80</td>
<td>0,82</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Sig95</td>
<td>0,44</td>
<td>0,60</td>
<td>0,65</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Sig95</td>
<td>0,35</td>
<td>0,53</td>
<td>0,57</td>
</tr>
<tr>
<td>MEstimate-DEL-Sig95</td>
<td>0,77</td>
<td>1,06</td>
<td>1,15</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Sig95</td>
<td>0,28</td>
<td>0,39</td>
<td>0,40</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Sig95</td>
<td>0,68</td>
<td>0,91</td>
<td>1,01</td>
</tr>
<tr>
<td>Correlation-DEL-Sig95</td>
<td>0,71</td>
<td>0,85</td>
<td>1,01</td>
</tr>
<tr>
<td>Konfiguration</td>
<td>Zeitverhalten in sec</td>
<td>Hypothesengröße in</td>
<td>Genauigkeit in %</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0%-Paket</td>
<td>5%-Paket</td>
<td>10%-Paket</td>
</tr>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
</tr>
<tr>
<td>Laplace-FIND-Sig99</td>
<td>0,28</td>
<td>0,45</td>
<td>0,43</td>
</tr>
<tr>
<td>Precision-FIND-Sig99</td>
<td>0,25</td>
<td>0,35</td>
<td>0,37</td>
</tr>
<tr>
<td>Accuracy-FIND-Sig99</td>
<td>0,44</td>
<td>0,58</td>
<td>0,64</td>
</tr>
<tr>
<td>WRAcc-FIND-Sig99</td>
<td>0,69</td>
<td>1,24</td>
<td>1,38</td>
</tr>
<tr>
<td>Kloesgen-FIND-Sig99</td>
<td>0,50</td>
<td>0,71</td>
<td>0,76</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-Sig99</td>
<td>0,39</td>
<td>0,57</td>
<td>0,62</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-Sig99</td>
<td>0,33</td>
<td>0,49</td>
<td>0,57</td>
</tr>
<tr>
<td>MEstimate-FIND-Sig99</td>
<td>0,70</td>
<td>1,01</td>
<td>1,11</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-Sig99</td>
<td>0,27</td>
<td>0,40</td>
<td>0,38</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-Sig99</td>
<td>0,59</td>
<td>0,85</td>
<td>0,99</td>
</tr>
<tr>
<td>Correlation-FIND-Sig99</td>
<td>0,63</td>
<td>0,80</td>
<td>0,86</td>
</tr>
<tr>
<td>Laplace-DEL-Sig99</td>
<td>0,28</td>
<td>0,43</td>
<td>0,44</td>
</tr>
<tr>
<td>Precision-DEL-Sig99</td>
<td>0,26</td>
<td>0,36</td>
<td>0,40</td>
</tr>
<tr>
<td>Accuracy-DEL-Sig99</td>
<td>0,45</td>
<td>0,61</td>
<td>0,71</td>
</tr>
<tr>
<td>WRAcc-DEL-Sig99</td>
<td>0,76</td>
<td>1,34</td>
<td>1,49</td>
</tr>
<tr>
<td>Kloesgen-DEL-Sig99</td>
<td>0,57</td>
<td>0,80</td>
<td>0,82</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-Sig99</td>
<td>0,44</td>
<td>0,61</td>
<td>0,65</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-Sig99</td>
<td>0,36</td>
<td>0,53</td>
<td>0,57</td>
</tr>
<tr>
<td>MEstimate-DEL-Sig99</td>
<td>0,77</td>
<td>1,07</td>
<td>1,15</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-Sig99</td>
<td>0,28</td>
<td>0,39</td>
<td>0,40</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-Sig99</td>
<td>0,67</td>
<td>0,92</td>
<td>1,01</td>
</tr>
<tr>
<td>Correlation-DEL-Sig99</td>
<td>0,71</td>
<td>0,85</td>
<td>1,01</td>
</tr>
</tbody>
</table>

Anhang B - 11: Ergebnisse Covering und IREP mit Significance Testing, Signifikanzlevel 99%
<table>
<thead>
<tr>
<th>Konfiguration</th>
<th>0%-Paket</th>
<th>5%-Paket</th>
<th>10%-Paket</th>
<th>0%-Paket</th>
<th>5%-Paket</th>
<th>10%-Paket</th>
<th>0%-Paket</th>
<th>5%-Paket</th>
<th>10%-Paket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplace</td>
<td>1,45</td>
<td>3,47</td>
<td>5,54</td>
<td>41,07</td>
<td>60,72</td>
<td>78,51</td>
<td>79,43</td>
<td>72,67</td>
<td>66,57</td>
</tr>
<tr>
<td>Precision</td>
<td>1,60</td>
<td>3,92</td>
<td>6,06</td>
<td>45,82</td>
<td>70,95</td>
<td>88,60</td>
<td>79,13</td>
<td>72,57</td>
<td>66,36</td>
</tr>
<tr>
<td>Accuracy</td>
<td>1,47</td>
<td>2,42</td>
<td>3,46</td>
<td>35,37</td>
<td>42,44</td>
<td>54,79</td>
<td>78,78</td>
<td>73,36</td>
<td>67,50</td>
</tr>
<tr>
<td>WRAcc</td>
<td>0,57</td>
<td>0,87</td>
<td>1,02</td>
<td>4,25</td>
<td>4,46</td>
<td>4,39</td>
<td>79,16</td>
<td>74,93</td>
<td>70,27</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>0,98</td>
<td>2,21</td>
<td>1,81</td>
<td>15,32</td>
<td>30,60</td>
<td>31,58</td>
<td>81,47</td>
<td>75,39</td>
<td>69,62</td>
</tr>
<tr>
<td>Kloesgen0.3</td>
<td>1,19</td>
<td>2,42</td>
<td>3,52</td>
<td>29,07</td>
<td>40,07</td>
<td>49,04</td>
<td>80,70</td>
<td>74,82</td>
<td>68,88</td>
</tr>
<tr>
<td>Kloesgen0.2</td>
<td>1,34</td>
<td>2,50</td>
<td>3,78</td>
<td>35,44</td>
<td>45,19</td>
<td>59,04</td>
<td>80,17</td>
<td>74,06</td>
<td>67,73</td>
</tr>
<tr>
<td>MEstimate</td>
<td>0,88</td>
<td>1,29</td>
<td>1,60</td>
<td>13,25</td>
<td>16,46</td>
<td>18,53</td>
<td>82,08</td>
<td>75,41</td>
<td>70,85</td>
</tr>
<tr>
<td>MEstimate 0.5</td>
<td>1,41</td>
<td>3,60</td>
<td>5,77</td>
<td>42,30</td>
<td>64,33</td>
<td>83,09</td>
<td>79,20</td>
<td>72,95</td>
<td>68,84</td>
</tr>
<tr>
<td>MEstimate 13.97</td>
<td>0,80</td>
<td>1,31</td>
<td>1,58</td>
<td>15,30</td>
<td>20,54</td>
<td>22,75</td>
<td>81,91</td>
<td>75,88</td>
<td>70,36</td>
</tr>
<tr>
<td>Correlation</td>
<td>0,89</td>
<td>1,28</td>
<td>1,60</td>
<td>14,42</td>
<td>18,72</td>
<td>22,07</td>
<td>80,46</td>
<td>75,52</td>
<td>69,97</td>
</tr>
<tr>
<td>Laplace-FIND-MDL</td>
<td>0,19</td>
<td>0,27</td>
<td>0,27</td>
<td>0,91</td>
<td>0,84</td>
<td>0,53</td>
<td>60,55</td>
<td>57,44</td>
<td>53,83</td>
</tr>
<tr>
<td>Precision-FIND-MDL</td>
<td>0,16</td>
<td>0,25</td>
<td>0,25</td>
<td>0,65</td>
<td>0,53</td>
<td>0,40</td>
<td>59,86</td>
<td>55,42</td>
<td>53,40</td>
</tr>
<tr>
<td>Accuracy-FIND-MDL</td>
<td>0,34</td>
<td>0,47</td>
<td>0,53</td>
<td>1,28</td>
<td>1,12</td>
<td>1,07</td>
<td>68,20</td>
<td>63,77</td>
<td>60,63</td>
</tr>
<tr>
<td>WRAcc-FIND-MDL</td>
<td>0,56</td>
<td>1,04</td>
<td>1,18</td>
<td>1,86</td>
<td>1,67</td>
<td>1,39</td>
<td>71,12</td>
<td>66,73</td>
<td>63,02</td>
</tr>
<tr>
<td>Kloesgen-FIND-MDL</td>
<td>0,32</td>
<td>0,52</td>
<td>0,57</td>
<td>1,44</td>
<td>1,32</td>
<td>1,21</td>
<td>66,88</td>
<td>63,08</td>
<td>59,71</td>
</tr>
<tr>
<td>Kloesgen0.3-FIND-MDL</td>
<td>0,25</td>
<td>0,40</td>
<td>0,46</td>
<td>1,04</td>
<td>1,12</td>
<td>0,79</td>
<td>64,35</td>
<td>61,20</td>
<td>58,11</td>
</tr>
<tr>
<td>Kloesgen0.2-FIND-MDL</td>
<td>0,21</td>
<td>0,36</td>
<td>0,39</td>
<td>1,00</td>
<td>0,93</td>
<td>0,61</td>
<td>63,14</td>
<td>59,31</td>
<td>56,53</td>
</tr>
<tr>
<td>MEstimate-FIND-MDL</td>
<td>0,37</td>
<td>0,57</td>
<td>0,66</td>
<td>1,44</td>
<td>1,33</td>
<td>1,11</td>
<td>66,24</td>
<td>62,72</td>
<td>59,31</td>
</tr>
<tr>
<td>MEstimate 0.5-FIND-MDL</td>
<td>0,17</td>
<td>0,27</td>
<td>0,27</td>
<td>0,75</td>
<td>0,68</td>
<td>0,47</td>
<td>60,27</td>
<td>56,24</td>
<td>53,41</td>
</tr>
<tr>
<td>MEstimate 13.97-FIND-MDL</td>
<td>0,35</td>
<td>0,55</td>
<td>0,63</td>
<td>1,30</td>
<td>1,23</td>
<td>1,09</td>
<td>65,14</td>
<td>61,25</td>
<td>58,64</td>
</tr>
<tr>
<td>Correlation-FIND-MDL</td>
<td>0,45</td>
<td>0,62</td>
<td>0,68</td>
<td>1,70</td>
<td>1,63</td>
<td>1,44</td>
<td>70,12</td>
<td>65,80</td>
<td>62,76</td>
</tr>
<tr>
<td>Laplace-DEL-MDL</td>
<td>0,20</td>
<td>0,30</td>
<td>0,28</td>
<td>0,93</td>
<td>0,79</td>
<td>0,58</td>
<td>61,46</td>
<td>57,88</td>
<td>54,44</td>
</tr>
<tr>
<td>Precision-DEL-MDL</td>
<td>0,18</td>
<td>0,27</td>
<td>0,29</td>
<td>0,74</td>
<td>0,58</td>
<td>0,49</td>
<td>60,72</td>
<td>56,29</td>
<td>53,76</td>
</tr>
<tr>
<td>Accuracy-DEL-MDL</td>
<td>0,33</td>
<td>0,46</td>
<td>0,53</td>
<td>1,25</td>
<td>1,04</td>
<td>0,96</td>
<td>67,73</td>
<td>63,41</td>
<td>60,41</td>
</tr>
<tr>
<td>WRAcc-DEL-MDL</td>
<td>0,57</td>
<td>1,08</td>
<td>1,23</td>
<td>1,84</td>
<td>1,68</td>
<td>1,37</td>
<td>70,69</td>
<td>66,21</td>
<td>62,68</td>
</tr>
<tr>
<td>Kloesgen-DEL-MDL</td>
<td>0,32</td>
<td>0,52</td>
<td>0,56</td>
<td>1,40</td>
<td>1,32</td>
<td>1,11</td>
<td>66,69</td>
<td>63,51</td>
<td>59,75</td>
</tr>
<tr>
<td>Kloesgen0.3-DEL-MDL</td>
<td>0,26</td>
<td>0,43</td>
<td>0,47</td>
<td>1,07</td>
<td>1,11</td>
<td>0,86</td>
<td>64,96</td>
<td>61,62</td>
<td>58,54</td>
</tr>
<tr>
<td>Kloesgen0.2-DEL-MDL</td>
<td>0,24</td>
<td>0,39</td>
<td>0,43</td>
<td>0,98</td>
<td>0,86</td>
<td>0,74</td>
<td>64,00</td>
<td>60,37</td>
<td>57,04</td>
</tr>
<tr>
<td>MEstimate-DEL-MDL</td>
<td>0,36</td>
<td>0,58</td>
<td>0,64</td>
<td>1,40</td>
<td>1,32</td>
<td>1,05</td>
<td>65,82</td>
<td>62,30</td>
<td>59,21</td>
</tr>
<tr>
<td>MEstimate 0.5-DEL-MDL</td>
<td>0,19</td>
<td>0,29</td>
<td>0,30</td>
<td>0,82</td>
<td>0,70</td>
<td>0,56</td>
<td>61,20</td>
<td>56,96</td>
<td>54,28</td>
</tr>
<tr>
<td>MEstimate 13.97-DEL-MDL</td>
<td>0,34</td>
<td>0,53</td>
<td>0,95</td>
<td>1,28</td>
<td>1,21</td>
<td>1,53</td>
<td>65,10</td>
<td>61,26</td>
<td>58,60</td>
</tr>
<tr>
<td>Correlation-DEL-MDL</td>
<td>0,44</td>
<td>0,60</td>
<td>0,67</td>
<td>1,70</td>
<td>1,56</td>
<td>1,35</td>
<td>69,32</td>
<td>65,37</td>
<td>61,86</td>
</tr>
</tbody>
</table>

Anhang B - 12: Ergebnisse Covering und IREP mit MDL-Kriterium
Anhang C – API-Dokumentation
In diesem Anhang findet sich die Dokumentation der API. In Tabelle Anhang C-1 sind die Pakete, deren Autoren und eine kurze Beschreibung der Pakete zu finden.

<table>
<thead>
<tr>
<th>Paket</th>
<th>Autor</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>seco</td>
<td>J. Fürnkranz, B. Werling</td>
<td>Enthält statistische Klassen für Regeln und Regelmengen</td>
</tr>
<tr>
<td>seco.heuristics</td>
<td>J. Fürnkranz, B. Werling</td>
<td>Enthält die Suchheuristiken</td>
</tr>
<tr>
<td>seco.learners</td>
<td>J. Fürnkranz</td>
<td>Enthält die Lernalgorithmen</td>
</tr>
<tr>
<td>seco.models</td>
<td>J. Fürnkranz</td>
<td>Enthält das Modell der Hypothesensprache</td>
</tr>
<tr>
<td>seco.pruning</td>
<td>B. Werling</td>
<td>Enthält die implementierten Pruningmechanismen</td>
</tr>
<tr>
<td>seco.pruning.criteria</td>
<td>B. Werling</td>
<td>Enthält die Stopkriterien</td>
</tr>
<tr>
<td>seco.pruning.model</td>
<td>B. Werling</td>
<td>Enthält das Modell der Pruningmechanismen</td>
</tr>
<tr>
<td>seco.pruning.operator</td>
<td>B. Werling</td>
<td>Enthält die Pruninhgoperatoren</td>
</tr>
</tbody>
</table>

Anhang C - 1: Übersicht über die Pakete

Klassen- und Interfaceverzeichnis

Class / Interface 1: RuleSetStats ................................................................. 101
Class / Interface 2: TwoClassStats ............................................................... 102
Class / Interface 3: Accuracy .............................................................................. 108
Class / Interface 4: CombinedMetaMultilayerPerceptron ...................................... 108
Class / Interface 5: Correlation .......................................................................... 109
Class / Interface 6: Cost_Measure ........................................................................ 109
Class / Interface 7: F_Measure ............................................................................. 110
Class / Interface 8: FoilGain ................................................................................ 111
Class / Interface 9: GeneralizedM ...................................................................... 112
Class / Interface 10: J_Measure .......................................................................... 113
Class / Interface 11: Kloesgen ........................................................................... 114
Class / Interface 12: Laplace ............................................................................... 115
Class / Interface 13: LineareCosts .................................................................... 115
Class / Interface 14: LinearRegression ............................................................... 116
Class / Interface 15: LinearRegression_Testversion ............................................. 117
Class / Interface 16: MEstimate .......................................................................... 118
Class / Interface 17: MetaMultilayerPerceptron ................................................... 119
Class / Interface 18: MetaSVMreg ...................................................................... 120
Class / Interface 19: MPrecision ......................................................................... 121
<table>
<thead>
<tr>
<th>Class / Interface</th>
<th>Class / Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>20: OptCandRuleMEs</td>
<td>21: OptCandRulePr</td>
</tr>
<tr>
<td>22: Precision</td>
<td>23: RateDiff</td>
</tr>
<tr>
<td>24: Relative_Cost_Measure</td>
<td>RIPPERPrune</td>
</tr>
<tr>
<td>26: RuleMDL</td>
<td>27: SearchHeuristic</td>
</tr>
<tr>
<td>28: TileRateDiff</td>
<td>29: WRAcc</td>
</tr>
<tr>
<td>30: Covering</td>
<td>31: GreedyTopDown</td>
</tr>
<tr>
<td>32: TopDownBeamSearch</td>
<td>Interface 33: CandidateRule</td>
</tr>
<tr>
<td>34: Condition</td>
<td>35: NominalCondition</td>
</tr>
<tr>
<td>36: NumericCondition</td>
<td>Interface 37: Rule</td>
</tr>
<tr>
<td>38: RuleSet</td>
<td>39: TrueCondition</td>
</tr>
<tr>
<td>39: ValueTestCondition</td>
<td>Interface 40: ValueTestCondition</td>
</tr>
<tr>
<td>40: REPOpt</td>
<td>41: IREPOpt</td>
</tr>
<tr>
<td>42: IREPruning</td>
<td>43: NoPruning</td>
</tr>
<tr>
<td>44: PrePruning</td>
<td>45: REPPruning</td>
</tr>
<tr>
<td>46: TDPruning</td>
<td>47: CutOff</td>
</tr>
<tr>
<td>48: MaximumErrorRate</td>
<td>Interface 49: MDL</td>
</tr>
<tr>
<td>50: NoStopping</td>
<td>51: RelativeCutOff</td>
</tr>
<tr>
<td>52: Significance</td>
<td>53: IRuleStoppingCriterion</td>
</tr>
<tr>
<td>54: IStoppingCriterion</td>
<td>Interface 55: Criterion</td>
</tr>
<tr>
<td>55: Criterion</td>
<td>56: PruningTemplate</td>
</tr>
<tr>
<td>57: RuleOperator</td>
<td>58: RuleSetOperator</td>
</tr>
<tr>
<td>59: RuleDeleteLastCondition</td>
<td>Interface 60: RuleDeleteLastCondition</td>
</tr>
<tr>
<td>60: RuleFindBestReplacement</td>
<td>Interface 61: RuleFindBestSimplification</td>
</tr>
<tr>
<td>61: RuleFindBestReplacement</td>
<td>Interface 62: RuleIdentity</td>
</tr>
<tr>
<td>62: RuleSetDeleteRule</td>
<td>Interface 63: RuleSetDeleteLastCondition</td>
</tr>
<tr>
<td>63: RuleFindBestSimplification</td>
<td>Interface 64: RuleSetDeleteRule</td>
</tr>
<tr>
<td>64: RuleSetDeleteRule</td>
<td>Interface 65: RuleSetDeleteRule</td>
</tr>
<tr>
<td>65: RuleSetSimplification</td>
<td>Interface 66: RuleSetIdentity</td>
</tr>
<tr>
<td>66: RuleSetIdentity</td>
<td>67: RuleSetSimplification</td>
</tr>
</tbody>
</table>

Vergleich von Pruningalgorithmen für Regellerner - Anhang
## C1 – Paket „seco“

<table>
<thead>
<tr>
<th>Klassen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleSetStats</td>
<td>This class counts the statistics (tp – TruePositives, fp – FalsePositives, fn – FalseNegatives, tn – TrueNegatives) for one ruleset for a given class.</td>
</tr>
<tr>
<td>TwoClassStats</td>
<td>Encapsulates performance functions for two-class problems.</td>
</tr>
</tbody>
</table>

### Class / Interface 1: RuleSetStats

public class RuleSetStats
extends java.lang.Object
implements java.io.Serializable
This class counts the statistics
tp - TruePositives
fp - FalsePositives
fn - FalseNegatives
tn - TrueNegatives
for one ruleset for a given class.

#### Constructor Detail

RuleSetStats
public RuleSetStats()  

#### Method Detail

updateCounts
public void updateCounts(RuleSet rs, 
weka.core.Instances data, 
double currClass)  
Updates the counts for a 2-class problem for the ruleset.

**Parameters:**
- rs - The ruleset to calculate the statistics for.
- data - The data on which the statistics are calculated
- currClass - The current class which is learned. The examples of this class count as tp.

toTwoClassStats
public TwoClassStats toTwoClassStats()  
This method returns a TwoClassStats Object with the tp,fp,fn,tn of the ruleset

**Returns:**
a TwoClassStats object with the tp,fp,fn,tn of the ruleset

toString
public java.lang.String toString()  
**Overrides:**
toString in class java.lang.Object
Class / Interface 2: TwoClassStats

public class TwoClassStats
extends java.lang.Object
implements java.lang.Cloneable, java.io.Serializable
Encapsulates performance functions for two-class problems.

Constructor Detail

TwoClassStats
public TwoClassStats()
Initializes a TwoClassStats with all 0s.

TwoClassStats
public TwoClassStats(double tp,
    double fp,
    double tn,
    double fn)
Creates the TwoClassStats with the given initial performance values.

Parameters:
  tp - the number of correctly classified positives
  fp - the number of incorrectly classified negatives
  tn - the number of correctly classified negatives
  fn - the number of incorrectly classified positives

Method Detail

clone
public java.lang.Object clone()
Overrides:
clone in class java.lang.Object
Returns:
a deep copy of the statistics

setTruePositive
public void setTruePositive(double tp)
Sets the number of positive instances predicted as positive

setFalsePositive
public void setFalsePositive(double fp)
Sets the number of negative instances predicted as positive

setTrueNegative
public void setTrueNegative(double tn)
Sets the number of negative instances predicted as negative

setFalseNegative
public void setFalseNegative(double fn)
Sets the number of positive instances predicted as negative

incTruePositive
public double incTruePositive()
Increments the number of true positives

incTruePositive
public double incTruePositive(double tp)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>incTrueNegative()</code></td>
<td>Increments the number of true negatives</td>
</tr>
<tr>
<td><code>incTrueNegative(tn)</code></td>
<td></td>
</tr>
<tr>
<td><code>incFalsePositive()</code></td>
<td>Increments the number of false positives</td>
</tr>
<tr>
<td><code>incFalsePositive(fp)</code></td>
<td></td>
</tr>
<tr>
<td><code>incFalseNegative()</code></td>
<td>Increments the number of false negatives</td>
</tr>
<tr>
<td><code>incFalseNegative(fn)</code></td>
<td></td>
</tr>
<tr>
<td><code>getTruePositive()</code></td>
<td>Gets the number of positive instances predicted as positive</td>
</tr>
<tr>
<td><code>getFalsePositive()</code></td>
<td>Gets the number of negative instances predicted as positive</td>
</tr>
<tr>
<td><code>getTrueNegative()</code></td>
<td>Gets the number of negative instances predicted as negative</td>
</tr>
<tr>
<td><code>getFalseNegative()</code></td>
<td>Gets the number of positive instances predicted as negative</td>
</tr>
<tr>
<td><code>getCorrect()</code></td>
<td>Gets the number of correctly predicted instances</td>
</tr>
<tr>
<td><code>getIncorrect()</code></td>
<td>Gets the number of incorrectly predicted instances</td>
</tr>
<tr>
<td><code>getIsPositive()</code></td>
<td>Gets the total number of positive instances</td>
</tr>
<tr>
<td><code>getIsNegative()</code></td>
<td>Gets the total number of negative instances</td>
</tr>
<tr>
<td><code>getPredictedPositive()</code></td>
<td></td>
</tr>
</tbody>
</table>
Gets the total number of examples predicted positive

getPredictedNegative
public double getPredictedNegative()
Gets the total number of examples predicted negative

getTotal
public double getTotal()
Gets the total number of examples

getErrorRate
public double getErrorRate()
Calculate the error rate. This is defined as
incorrectly classified examples
--------------------------------------------------------
total examples

Returns:
the error rate

getAccuracy
public double getAccuracy()
Calculate the accuracy. This is defined as
correctly classified examples
--------------------------------------------------------
total examples

Returns:
the accuracy

getPrior
public double getPrior()
Estimate the prior probability of positive examples. This is defined as
positive examples
-----------------
total examples

Returns:
the prior

getTruePositiveRate
public double getTruePositiveRate()
Calculate the true positive rate. This is defined as
correctly classified positives
--------------------------------
total positives

Returns:
the true positive rate

getFalsePositiveRate
public double getFalsePositiveRate()
Calculate the false positive rate. This is defined as
incorrectly classified negatives
--------------------------------
total negatives

**Returns:**
the false positive rate

---

getPrecision
public double getPrecision()
Calculate the precision. This is defined as
\[
\frac{\text{correctly classified positives}}{\text{total predicted as positive}}
\]

**Returns:**
the precision

---

getRecall
public double getRecall()
Calculate the recall. This is defined as
\[
\frac{\text{correctly classified positives}}{\text{total positives}}
\]
(Which is also the same as the truePositiveRate.)

**Returns:**
the recall

---

getFMeasure
public double getFMeasure()
Calculate the F-Measure. This is defined as
\[
\frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}
\]

**Returns:**
the F-Measure

---

getFallout
public double getFallout()
Calculate the fallout. This is defined as
\[
\frac{\text{incorrectly classified negatives}}{\text{total predicted as positive}}
\]

**Returns:**
the fallout

---

getConfusionMatrix
public weka.classifiers.evaluation.ConfusionMatrix getConfusionMatrix()
Generates a ConfusionMatrix representing the current two-class statistics, using class names "negative" and "positive".

**Returns:**
a ConfusionMatrix.

---

toString
public java.lang.String toString()
Returns a string containing the various performance measures for the current object

Overrides:

toString in class java.lang.Object
<table>
<thead>
<tr>
<th>Klasse</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Correlation</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Cost Measure</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>F Measure</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>FoilGain</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>GeneralizedM</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>J Measure</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Kloesgen</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Laplace</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>LinearCosts</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>LinearRegression</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>LinearRegression_Testversion</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>MEstimate</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>MetaMultilayerPerceptron</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>MetaSVMreg</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>MPrecision</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>OptCandRuleMEstimate</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>OptCandRulePrec</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Precision</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>RateDiff</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Relative_Cost_Measure</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>RIPPERPrune</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>RuleMDL</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>SearchHeuristic</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>TileRateDiff</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>WRAcc</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.120</td>
</tr>
</tbody>
</table>
Class / Interface 3: Accuracy

public class Accuracy
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with accuracy. The accuracy of a single rule is \((tp + tn) / (tp + fp + fn + tn)\).
You may also want to consider the equivalent but faster Difference. For more details on the the equivalences between search heuristics see (Fürnkranz & Flach, ICML-03).

Constructor Detail

Accuracy
public Accuracy()
Constructor

Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
return the accuracy of the rule

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule

Class / Interface 4: CombinedMetaMultilayerPerceptron

public class CombinedMetaMultilayerPerceptron
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

Constructor Detail

CombinedMetaMultilayerPerceptron
public CombinedMetaMultilayerPerceptron()
throws java.lang.Exception

Throws:
java.lang.Exception

Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule

getOptions
public java.lang.String[] getOptions()

**Specified by:**
getOptions in interface weka.core.OptionHandler

---

listOptions
public java.util.Enumeration listOptions()

**Specified by:**
listOptions in interface weka.core.OptionHandler

---

setOptions
public void setOptions(java.lang.String[] options)

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Throws:**
java.lang.Exception

---

### Class / Interface 5: Correlation

public class Correlation
extends SearchHeuristic
implements java.io.Serializable

The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with the Correlation estimate, i.e. \( \frac{tp*tn - fp*fn}{\sqrt{Pos*Neg*Covered*Uncovered}} \)

#### Constructor Detail

Correlation
public Correlation()

#### Method Detail

evaluateRule
double evaluateRule(CandidateRule r)
evaluates a rule with the Correlation estimate

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

---

### Class / Interface 6: Cost_Measure

class Cost_Measure
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

#### Constructor Detail

Cost_Measure
public Cost_Measure()
### Cost_Measure

```java
public Cost_Measure(double c)
    throws java.lang.Exception

Throws:
java.lang.Exception
```

### Method Detail

evaluateRule

```java
public double evaluateRule(CandidateRule r)
```

**Description copied from class: SearchHeuristic**

computes the evaluation for a possible rule.

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

### listOptions

```java
public java.util.Enumeration listOptions()
```

**Specified by:**
listOptions in interface weka.core.OptionHandler

### setOptions

```java
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
```

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Throws:**
java.lang.Exception

### getOptions

```java
public java.lang.String[] getOptions()
```

**get the current configuration**

**Specified by:**
getOptions in interface weka.core.OptionHandler

**Returns:**
an array of strings suitable for passing to setOptions

---

### Class / Interface 7: F_Measure

```java
public class F_Measure
    extends SearchHeuristic
    implements weka.core.OptionHandler, java.io.Serializable
```

The seco package implements generic functionality for simple separate-and-conquer rule learning.
This file implements a generic class for evaluating a rule with F-Measure.

### Constructor Detail

**F_Measure**

```java
public F_Measure()
```

**F_Measure**

```java
public F_Measure(double n)
```
### Method Detail

**evaluateRule**

```java
public double evaluateRule(CandidateRule r)
```

Evaluates the F-Measure of the rule

**Parameters:**

- `r` - the candidate rule

**Throws:**

`java.lang.Exception`

### Constructor Detail

**FoilGain**

```java
public FoilGain()
```

The `sec` package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with Foil's information gain, i.e. \(\text{tp} \times \left(\log_2\left(\frac{\text{tp}}{\text{tp} + \text{fp}}\right) - \log_2\left(\frac{\text{tp}'}{\text{tp}' + \text{fp}'}\right)\right)\) where `tp` and `tp'` are the true and false positives of the parent rule.

**Method Detail**

**evaluateRule**

```java
public double evaluateRule(CandidateRule r)
```

### Class / Interface 8: FoilGain

**public class FoilGain**

extends SearchHeuristic

implements java.io.Serializable

The `sec` package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with Foil's information gain, i.e. \(\text{tp} \times \left(\log_2\left(\frac{\text{tp}}{\text{tp} + \text{fp}}\right) - \log_2\left(\frac{\text{tp}'}{\text{tp}' + \text{fp}'}\right)\right)\) where `tp` and `tp'` are the true and false positives of the parent rule.
evaluates a rule with the Foil's information gain heuristic. Rules without predecessors (getPredecessor() == null) are evaluated with 0.

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
- r - the candidate rule

### Class / Interface 9: GeneralizedM

```java
class GeneralizedM extends SearchHeuristic
  implements weka.core.OptionHandler, java.io.Serializable

The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with the generalized m-estimate, i.e. 
\[(tp+m*c)/(tp+fp+m)\].

c may be interpreted as a general linear cost factor, just like in LinearCosts.
m is the m-value as in the m-heuristic. It may be viewed as a trade-off between LinearCost (which assumes a cost value c) and Precision (which does not make any cost assumptions).
If m is NaN, it will be interpreted as infinity and the LinearCosts heuristic will be called. If m is 0, you get Precision.
The default values are m = 2 and c = 0.5, which results in the Laplace heuristic.
See (Fürnkranz & Flach, ICML-03) for details on the Generalized MEstimate.
```

#### Constructor Detail

**GeneralizedM**

```java
public GeneralizedM()

Empty constructor, c will be set to 0.5 and m to 2.0.
```

**GeneralizedM**(double m, double c) throws java.lang.Exception

**Parameters:**
- m - the value for m (0 <= m <= NaN, default 1)
- c - the cost factor (0 <= c <= 1, default 0.5)

**Throws:**
java.lang.Exception

#### Method Detail

**evaluateRule**

```java
public double evaluateRule(CandidateRule r)

evaluates a rule with the generalized m-estimate
```

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
- r - the candidate rule

**listOptions**

```java
public java.util.Enumeration listOptions()

returns an enumeration of the available options, which are: -M number
The m-value used in the m-heuristic (Default: 2).
```
The cost trade-off in the heuristic. 0 <= c <= 1 (Default: 0.5).

Specified by:
listOptions in interface weka.core.OptionHandler

setOptions
public void setOptions(java.lang.String[] options)
        throws java.lang.Exception
parse a list of options.

Specified by:
setOptions in interface weka.core.OptionHandler

Parameters:
options - the list of options as an array of strings

Throws:
java.lang.Exception - if an option is not supported

getOptions
public java.lang.String[] getOptions()
get the current configuration

Specified by:
getOptions in interface weka.core.OptionHandler

Returns:
an array of strings suitable for passing to setOptions

Class / Interface 10: J_Measure
public class J_Measure
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning.
This file implements a generic class for evaluating a rule with the kloesgen-estimate. The default value
of n is 2. It can be changed via setOptions (the class implements the OptionHandler interface).

Constructor Detail

J_Measure
public J_Measure()

Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule
### Class / Interface 11: Kloesgen

public class **Kloesgen**
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with the kloesgen-estimate. The default value of n is 2. It can be changed via setOptions (the class implements the OptionHandler interface).

#### Constructor Detail

**Kloesgen**

public **Kloesgen**()  

**Kloesgen**

public **Kloesgen**(double n)

**Throws:**
java.lang.Exception

#### Method Detail

**evaluateRule**

public double **evaluateRule**(CandidateRule r)

*Description copied from class: SearchHeuristic*

computes the evaluation for a possible rule.

*Specified by:*  
evaluateRule in class SearchHeuristic

*Parameters:*  
r - the candidate rule

**listOptions**

public java.util.Enumeration **listOptions**()

returns an enumeration of the available options, which are: -M number
The n-value used in the Kloesgen-heuristic (Default: 2.0).

*Specified by:*  
listOptions in interface weka.core.OptionHandler

**setOptions**

public void **setOptions**(java.lang.String[] options)

*Throws:*  
java.lang.Exception

*Specified by:*  
setOptions in interface weka.core.OptionHandler

**getOptions**

public java.lang.String[] **getOptions**()

get the current configuration

*Specified by:*  
getOptions in interface weka.core.OptionHandler

*Returns:*  
an array of strings suitable for passing to setOptions
Class / Interface 12: Laplace

public class Laplace
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with the Laplace-estimate, i.e. 
\( \frac{tp+1}{tp+fp+2} \)

Constructor Detail

Laplace()

Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
evaluates a rule with the Laplace estimate

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule

Class / Interface 13: LinearCosts

public class LinearCosts
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. Linear Costs implements a class for evaluating a rule with a linear cost function \( c*tp - (1-c)*fp \), where \( 0 \leq c \leq 1 \). 
\( c = 1 \) means that only covered positives counts, \( c = 0 \) means that only excluding negatives counts, values inbetween trade off between these two extremes. The default value of \( c \) is 0.5, which corresponds to the Accuracy heuristic.
The parameter can be changed via setOptions (the class implements the OptionHandler interface).

Constructor Detail

LinearCosts()
Empty constructor, \( c \) will be set to 0.5.

LinearCosts(double c)
throws java.lang.Exception
Constructor.

Parameters:
c - the cost trade-off.

Throws:
java.lang.Exception - unless 0 <= c <= 1.

**Method Detail**

evaluateRule
public double **evaluateRule**(CandidateRule r)
evaluates a rule with a linear cost metric
**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

listOptions
public java.utilEnumeration **listOptions**()
returns an enumeration of the available options. which are: -c number
The cost trade-off in the heuristic. 0 <= c <= 1 (Default: 0.5).
**Specified by:**
listOptions in interface weka.core.OptionHandler

setOptions
public void **setOptions**(java.lang.String[] options)
throws java.lang.Exception
parse a list of options.
**Specified by:**
setOptions in interface weka.core.OptionHandler

**Parameters:**
options - the list of options as an array of strings

**Throws:**
java.lang.Exception - if an option is not supported

getOptions
public java.lang.String[] **getOptions**()
get the current configuration
**Specified by:**
getOptions in interface weka.core.OptionHandler

**Returns:**
an array of strings suitable for passing to setOptions

**Class / Interface 14: LinearRegression**
public class **LinearRegression**
extends SearchHeuristic
implements java.io.Serializable

**Constructor Detail**

LinearRegression
public **LinearRegression**()

**Method Detail**
evaluateRule

public double `evaluateRule`(CandidateRule r)

Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

Specified by:
`evaluateRule` in class SearchHeuristic

Parameters:
- r - the candidate rule

Class / Interface 15: LinearRegression_Testversion
public class `LinearRegression_Testversion`
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

Constructor Detail

LinearRegression_Testversion
public `LinearRegression_Testversion`()

LinearRegression_Testversion
public `LinearRegression_Testversion`(double[] values)
throws java.lang.Exception

Throws:
java.lang.Exception

Method Detail

evaluateRule
public double `evaluateRule`(CandidateRule r)

Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

Specified by:
`evaluateRule` in class SearchHeuristic

Parameters:
- r - the candidate rule

setValues
public void `setValues`(double[] values)
throws java.lang.Exception

Throws:
java.lang.Exception

listOptions
public java.util.Enumeration `listOptions`()

Specified by:
`listOptions` in interface weka.core.OptionHandler

setOptions
public void `setOptions`(java.lang.String[] options)
throws java.lang.Exception

Specified by:
`setOptions` in interface weka.core.OptionHandler

Throws:
```java
java.lang.Exception
getOptions
public java.lang.String[] getOptions()
get the current configuration
**Specified by:**
getOptions in interface weka.core.OptionHandler
**Returns:**
an array of strings suitable for passing to setOptions
```

---

### Class / Interface 16: MEstimate

A public class `MEstimate` extends `SearchHeuristic` and implements `weka.core.OptionHandler` and `java.io.Serializable`. This package implements generic functionality for simple separate-and-conquer rule learning. A file implements a generic class for evaluating a rule with the m-estimate, i.e. 
\[
\frac{tp+m \cdot prior}{tp+fp+m}.
\]
The prior probability is 
\[
\frac{tp + fn}{tp + fp + fn + tn}.
\]
If you don't want to recompute this every time around, better use the `GeneralizedM`. The default value of `m` is 2. It can be changed via `setOptions` (the class implements the `OptionHandler` interface).

#### Constructor Detail

**MEstimate**

```java
public MEstimate()
Empty constructor, m will be set to 1.
```

**MEstimate**

```java
public MEstimate(double m)
throws java.lang.Exception
```

**Parameters:**
m - the value for `m`

**Throws:**
`java.lang.Exception`

#### Method Detail

**evaluateRule**

```java
public double evaluateRule(CandidateRule r)
evaluates a rule with the m-estimate
**Specified by:**
evaluateRule in class SearchHeuristic
**Parameters:**
r - the candidate rule
```

**listOptions**

```java
public java.util.Enumeration listOptions()
returns an enumeration of the available options, which are: -M number
The m-value used in the m-heuristic (Default: 2).
**Specified by:**
listOptions in interface weka.core.OptionHandler
```
setOptions
public void setOptions(java.lang.String[] options)
 throws java.lang.Exception
parse a list of options.
**Specified by:**
setOptions in interface weka.core.OptionHandler
**Parameters:**
options - the list of options as an array of strings
**Throws:**
java.lang.Exception - if an option is not supported

getOptions
public java.lang.String[] getOptions()
get the current configuration
**Specified by:**
getOptions in interface weka.core.OptionHandler
**Returns:**
an array of strings suitable for passing to setOptions

### Class / Interface 17: MetaMultilayerPerceptron

public class MetaMultilayerPerceptron
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

#### Constructor Detail

MetaMultilayerPerceptron
public MetaMultilayerPerceptron()

MetaMultilayerPerceptron
public MetaMultilayerPerceptron(java.lang.String ModelName)
 throws java.lang.Exception

**Throws:**
java.lang.Exception

#### Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
**Description copied from class: SearchHeuristic**
computes the evaluation for a possible rule.
**Specified by:**
evaluateRule in class SearchHeuristic
**Parameters:**
r - the candidate rule

listOptions
public java.util.Enumeration listOptions()
**Specified by:**
listOptions in interface weka.core.OptionHandler
setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
Specified by:
setOptions in interface weka.core.OptionHandler
Throws:
java.lang.Exception

getOptions
public java.lang.String[] getOptions()
get the current configuration
Specified by:
getOptions in interface weka.core.OptionHandler
Returns:
an array of strings suitable for passing to setOptions

Class / Interface 18: MetaSVMreg

public class MetaSVMreg
extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

Constructor Detail
MetaSVMreg
public MetaSVMreg()

MetaSVMreg
public MetaSVMreg(java.lang.String ModelName)
    throws java.lang.Exception
Throws:
java.lang.Exception

Method Detail
evaluateRule
public double evaluateRule(CandidateRule r)
Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.
Specified by:
evaluateRule in class SearchHeuristic
Parameters:
r - the candidate rule

getOptions
public java.lang.String[] getOptions()
Specified by:
getOptions in interface weka.core.OptionHandler

listOptions
public java.util.Enumeration listOptions()
Specified by:
listOptions in interface weka.core.OptionHandler
```java
public void setOptions(String[] options)
    throws Exception

Specified by:
setOptions in interface weka.core.OptionHandler

Throws:
java.lang.Exception
```

---

**Class / Interface 19: MPrecision**

```java
public class MPrecision
    extends SearchHeuristic
    implements weka.core.OptionHandler, java.io.Serializable
```

**Constructor Detail**

```java
public MPrecision()
```

```java
public MPrecision(double c)
    throws Exception
```

**Throws:**

java.lang.Exception

---

**Method Detail**

```java
evaluateRule
public double evaluateRule(CandidateRule r)

Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule
```

```java
listOptions
public java.util.Enumeration listOptions()

Specified by:
listOptions in interface weka.core.OptionHandler
```

```java
setOptions
public void setOptions(String[] options)
    throws Exception

Specified by:
setOptions in interface weka.core.OptionHandler

Throws:
java.lang.Exception
```

```java
getOptions
public java.lang.String[] getOptions()

get the current configuration

Specified by:
getOptions in interface weka.core.OptionHandler

Returns:
```
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file provides evaluation for evaluating a rule with precision, i.e. $\frac{tp}{(tp+fp)}$.

### Class / Interface 20: OptCandRuleMEstimate

**public class** `OptCandRuleMEstimate`

**extends** `SearchHeuristic`

**implements** `java.io.Serializable`

The seco package implements generic functionality for simple separate-and-conquer rule learning. This file provides evaluation for evaluating a rule with precision, i.e. $\frac{tp}{(tp+fp)}$.

#### Constructor Detail

`OptCandRuleMEstimate`

**public** `OptCandRuleMEstimate()`

#### Method Detail

**evaluateRule**

`public double evaluateRule(CandidateRule r)`

evaluates the refinement of a rule by trading off the precision and WRA which both got their origin in the point (N,P) - that's because in this case refinements are evaluated with a Top-Down Strategy.

**Specified by:**
evaluateRule in class `SearchHeuristic`

**Parameters:**

- `r` - the candidate rule

**listOptions**

`public java.util.Enumeration listOptions()`

returns an enumeration of the available options, which are: -M number

The m-value used in the m-heuristic (Default: 2).

**setOptions**

`public void setOptions(java.lang.String[] options)`

throws `java.lang.Exception`

parse a list of options.

**Parameters:**

- `options` - the list of options as an array of strings

**Throws:**

- `java.lang.Exception` - if an option is not supported

**getOptions**

`public java.lang.String[] getOptions()`

get the current configuration

**Returns:**

an array of strings suitable for passing to `setOptions`

---

### Class / Interface 21: OptCandRulePrec

**public class** `OptCandRulePrec`

**extends** `SearchHeuristic`

**implements** `java.io.Serializable`
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file provides evaluation for evaluating a rule with precision, i.e. \( \frac{tp}{tp+fp} \)

**Constructor Detail**

OptCandRulePrec
public **OptCandRulePrec**()

**Method Detail**
evaluateRule
public double **evaluateRule**(CandidateRule r)
evaluates the precision of a candidate refinement; this works by maximizing the angle between the (P-p)-axis and the (N-n)-axis; thus, if a rule is successive specialized starting with the most general rule, the specialization which is located nearest to the (N-n)-axis is chosen

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
- r - the candidate rule

---

**Class / Interface 22: Precision**

public class **Precision**
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file provides evaluation for evaluating a rule with precision, i.e. \( \frac{tp}{tp+fp} \)

**Constructor Detail**

Precision
public **Precision**()

**Method Detail**
evaluateRule
public double **evaluateRule**(CandidateRule r)
evaluates the precision of the rule

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
- r - the candidate rule

---

**Class / Interface 23: RateDiff**

public class **RateDiff**
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. This file implements a generic class for evaluating a rule with the difference of the true positive rate and the true negative rate. For rules with the same example distribution ((tp + fn) and (fp + tn) are
constant), this is equivalent to weighted relative accuracy (WRAcc), but presumably faster.

**Constructor Detail**

RateDiff
public RateDiff()
Constructor

**Method Detail**

evaluateRule
public double evaluateRule(CandidateRule r)
evaluates a rule with weighted relative accuracy

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

---

**Class / Interface 24: Relative_Cost_Measure**

public class Relative_Cost_Measure extends SearchHeuristic
implements weka.core.OptionHandler, java.io.Serializable

**Constructor Detail**

Relative_Cost_Measure
public Relative_Cost_Measure()

Relative_Cost_Measure
public Relative_Cost_Measure(double c)
throws java.lang.Exception

**Throws:**
java.lang.Exception

**Method Detail**

evaluateRule
public double evaluateRule(CandidateRule r)

**Description copied from class: SearchHeuristic**
computes the evaluation for a possible rule.

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

listOptions
public java.util.Enumeration listOptions()

**Specified by:**
listOptions in interface weka.core.OptionHandler

setOptions
public void setOptions(java.lang.String[] options)

---
Vergleich von Pruningalgorithmen für Regellerner - Anhang

---

throws java.lang.Exception

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Throws:**
java.lang.Exception

getOptions
public java.lang.String[] getOptions()
get the current configuration

**Specified by:**
getOptions in interface weka.core.OptionHandler

**Returns:**
an array of strings suitable for passing to setOptions

---

**Class / Interface 25: RIPPERPrune**
Evaluates a rule with the Ripper pruning heuristic.

\[ h(n, p) = \frac{p - n}{p + n} \]

**Constructor Detail**

RIPPERPrune
public RIPPERPrune()

**Method Detail**
evaluateRule
public double evaluateRule(CandidateRule r)

Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

---

**Class / Interface 26: RuleMDL**
public class RuleMDL
extends SearchHeuristic
implements java.io.Serializable

MDL based Cost Measure based on Bernd Pfahringer’s MDL formula.

**Constructor Detail**

RuleMDL
public RuleMDL()

**Method Detail**
evaluateRule
public double evaluateRule(CandidateRule r)
Description copied from class: SearchHeuristic
computes the evaluation for a possible rule.

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule

<table>
<thead>
<tr>
<th>Constructor Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchHeuristic</td>
</tr>
<tr>
<td>public SearchHeuristic()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluateRule</td>
</tr>
<tr>
<td>public abstract double evaluateRule(CandidateRule r)</td>
</tr>
<tr>
<td>computes the evaluation for a possible rule.</td>
</tr>
</tbody>
</table>

| Parameters: |
| r - the candidate rule |

| forName |
| public static SearchHeuristic forName(java.lang.String name, java.lang.String[] options) |
| throws java.lang.Exception |
| Creates a new instance of the heuristic given it's class name and (optional) arguments to pass to it's setOptions method. If the heuristic implements OptionHandler and the options parameter is non-null, the heuristic will have it's options set. |

| Parameters: |
| name - the fully qualified class name of the heuristic |
| options - an array of options suitable for passing to setOptions. May be null. |

| Returns: |
| the newly created heuristic, ready for use. |

| Throws: |
| java.lang.Exception - if the classifier name is invalid, or the options supplied are not acceptable to the classifier |

| toOptionString |
| public java.lang.String toOptionString() |
| Returns: |
| a string representation of a search heuristic, including all parameters (if any). If there are parameters, the returned string will start and end with quotes. Thus the representation is suitable for the command- |
line (e.g., for initializing other objects).

toString
public java.lang.String toString()

Overrides:
toString in class java.lang.Object

Returns:
a string representation of a search heuristic, including all parameters (if any). The parameters are
added in parentheses, like a constructor call.

Class / Interface 28: TileRateDiff
public class TileRateDiff
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning.
This file implements a generic class for evaluating a rule with the difference of the true positive rate
and the true negative rate, multiplied by the size of the rules. This is motivated by Bart Goethals’ idea
of tiles as an interestingness measure for rules.

Constructor Detail

TileRateDiff
public TileRateDiff()
Constructor

Method Detail

evaluateRule
public double evaluateRule(CandidateRule r)
evaluates a rule with weighted relative accuracy

Specified by:
evaluateRule in class SearchHeuristic

Parameters:
r - the candidate rule

Class / Interface 29: WRAcc
public class WRAcc
extends SearchHeuristic
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning.
This file implements a generic class for evaluating a rule with weighted relative accuracy. weighted
relative accuracy is (tp+fp)/total (tp/(tp+fp) - prior) where total is tp+fp+fn+tn and prior is the prior
probability (tp+fn)/total.
You may also want to consider the equivalent but faster RateDiff. For more details on the the
 equivalences between search heuristics see (Fürnkranz & Flach, ICML-03).

Constructor Detail
<table>
<thead>
<tr>
<th>Method Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>constructor</strong></td>
</tr>
<tr>
<td>public <strong>WRAcc</strong>()</td>
</tr>
<tr>
<td><strong>evaluateRule</strong>(CandidateRule r)</td>
</tr>
</tbody>
</table>
evaluates a rule with weighted relative accuracy

**Specified by:**
evaluateRule in class SearchHeuristic

**Parameters:**
r - the candidate rule
C3 – Paket „seco.learners“

<table>
<thead>
<tr>
<th>Klasse</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covering</td>
<td>The seco package implements generic functionality for simple separate-</td>
</tr>
<tr>
<td></td>
<td>and-conquer rule learning.</td>
</tr>
<tr>
<td>GreedyTopDown</td>
<td>The seco package implements generic functionality for simple separate-</td>
</tr>
<tr>
<td></td>
<td>and-conquer rule learning.</td>
</tr>
<tr>
<td>TopDownBeamSearch</td>
<td>The seco package implements generic functionality for simple separate-</td>
</tr>
<tr>
<td></td>
<td>and-conquer rule learning.</td>
</tr>
</tbody>
</table>

Anhang C - 4: Paketinhalte des Pakets “seco.learners”

Class / Interface 30: Covering

```java
class Covering extends weka.classifiers.Classifier
implements weka.core.WeightedInstancesHandler, weka.core.AdditionalMeasureProducer,
weka.core.OptionHandler, java.io.Serializable
```

The seco package implements generic functionality for simple separate-and-conquer rule learning. Covering implements the covering strategy. It relies on external methods for learning a single rule. The covering strategy learns one rule at a time, each time removing all examples covered by the rule from the training set. The current implementation only uses the simplest stopping criterion: Learning stops whenever the best rule found leads to a decrease in accuracy on the training set (i.e., when the best rule does not cover more positive than negative examples). The class is intended as an outer loop for separate-and-conquer rule learning algorithms, but in principle other classifiers could be used as internal classifiers as well (this only makes sense if the classifier is partial, i.e., if it returns the missing class value for some of the instances). Currently, however, it requires a method for returning the learned model, which is not supported by the general Classifier class. So for the moment, the internal classifier has to be of type Rule, and the learner needs to be of type TopDown BeamSearch. TODO: - maybe be more specific upon which classifier is valid as the single-rule learner (must be one that does not classify all examples). The internal learner currently must be TopDownBeamSearch. This should be a generic class (like a "PartialClassifier") that has certain methods defined.

Constructor Detail

Covering
public Covering()

Method Detail

getRuleLearner
public TopDownBeamSearch getRuleLearner()

Returns:
the current rule learner

setRuleLearner
public void setRuleLearner(TopDownBeamSearch l)
specify the learner that will be used for inducing single rules

Parameters:
1 - the new rule learner (must be a Rule)

getRuleSet
public RuleSet getRuleSet()

Returns:
the learned set of rules
classifyInstance
public double classifyInstance(weka.core.Instance inst)
    throws java.lang.Exception
classify the passed Instance, i.e. return the class value if the instance is covered by the rule, or the missing value if it is not covered by the rule. Currently we assume classification as a decision list, i.e., the prediction of the first rule that doesn't predict the missing value is returned. Eventually, this should probably be a parameter or maybe even a separate subclass.

Overrides:
classifyInstance in class weka.classifiers.Classifier

Parameters:
inst - the instance

Returns:
the class of the instance or a missing value

Throws:
java.lang.Exception

buildClassifier
public void buildClassifier(weka.core.Instances instances)
    throws java.lang.Exception
build a classifier using the covering strategy. The covering strategy calls a partial classifier on the training data, removes all examples that are covered by it from the dataset, and repeats until no progress is made (i.e., until no more rules are learned.

Specified by:
buildClassifier in class weka.classifiers.Classifier

Throws:
java.lang.Exception

toString
public java.lang.String toString()

Overrides:
toString in class java.lang.Object

listOptions
public java.util.Enumeration listOptions()

Returns an enumeration describing the available options. Valid options are:

-D
turn on the debug mode and produce a lot of output
-L rule learner specify the internal rule learner (default: TopDownBeamSearch) --
option separator. All options after this will be passed to the rule learner.

Specified by:
listOptions in interface weka.core.OptionHandler

Overrides:
listOptions in class weka.classifiers.Classifier

Returns:
an enumeration of all available options

setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
parse a list of options. The Options for Pruning must come before the Learneroptions

Specified by:
setOptions in interface weka.core.OptionHandler

Overrides:
setOptions in class weka.classifiers.Classifier
**Parameters:**
opts - the list of options an array of strings

**Throws:**
Exception - if an option is not supported
java.lang.Exception

---

```java
public java.lang.String[] getOptions()
```
the current settings of the classifier

**Specified by:**
getOptions in interface weka.core.OptionHandler

**Overrides:**
getOptions in class weka.classifiers.Classifier

**Returns:**
an array of strings suitable for passing to setOptions

---

```java
public java.util Enumeration enumerateMeasures()
```

**Specified by:**
enumerateMeasures in interface weka.core.AdditionalMeasureProducer

**Returns:**
an enumeration of the additional measure names

---

```java
public double getMeasure(java.lang.String additionalMeasureName)
```

**Specified by:**
getMeasure in interface weka.core.AdditionalMeasureProducer

**Parameters:**
measureName - the name of the measure to query for its value

**Returns:**
the value of the named measure

**Throws:**
java.lang.IllegalArgumentException - if the named measure is not supported

---

```java
public static void main(java.lang.String[] args)
```
Main method.

---

```java
public PruningTemplate getPruner()
```

**Returns:**
the m_prune

---

```java
public void setPruner(PruningTemplate m_prune)
```

**Parameters:**
m_prune - the m_prune to set
### Class / Interface 31: GreedyTopDown

```java
public class GreedyTopDown extends weka.classifiers.Classifier
    implements weka.core.WeightedInstancesHandler, weka.core.AdditionalMeasureProducer, weka.core.OptionHandler
```

The seco package implements generic functionality for simple separate-and-conquer rule learning. GreedyTopDown implements the greedy top-down (general-to-specific) induction of a single rule. In principle, a Rule can be used as a classifier in its own right, but the main purpose of this is to provide a learner for the inner loop of the separate-and-conquer/covering loop. TODO: - Stopping heuristics - Filtering heuristics (at the place where we are now " ---> ignored") - maybe initialization with a given rule (be careful not to re-initialize the rule, init used vector) - beam search - local optimization (gain heuristics do not allow a global comparison between rules, such as Foil-Gain) - LEFs (multiple evaluation heuristics) - cleaner interface should maybe store a Rule (and not a CandidateRule) internally and copy it to a CandidateRule for learning - split value can be chosen from domain or interpolate between neighboring values - Maybe the search should maintain a general confusion matrix so that CN2 can implemented (both, the entropy heuristic and the decision list search that auto-picks the class) - have I covered missing class values? - the two code blocks for finding the best condition should probably be separate routines/methods of some class. Parts of it is based on code for JRip and for Prism.

#### Constructor Detail

**GreedyTopDown**

```java
public GreedyTopDown()
```

#### Method Detail

**initRule**

```java
public void initRule()
```

initialize the rule learner with a fresh, empty rule.

```java
public void initRule(CandidateRule r)
```

initialize the rule learner with

**Parameters:**

- `rule` - a new starting point for the refinement

**Returns:**

- the learned rule

**getHeuristic**

```java
public SearchHeuristic getHeuristic()
```

return the heuristic used for evaluating rules

**Returns:**

- a seco.heuristics.SearchHeuristic

**setHeuristic**

```java
public void setHeuristic(SearchHeuristic h)
```

set the heuristic used for evaluating rules

**Parameters:**

- `h` - a seco.heuristics.SearchHeuristic

**getTargetClass**
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public double getTargetClass()</code></td>
<td>Returns: the index of the class for which a rule is learned</td>
</tr>
<tr>
<td><code>public void setTargetClass(double i)</code></td>
<td>set the index of the class to a new value. It also resets the rule (initRule).</td>
</tr>
<tr>
<td><code>public boolean debug()</code></td>
<td>Returns: true if debugging is on, false if not</td>
</tr>
<tr>
<td><code>public void setDebug(boolean d)</code></td>
<td>Overrides: setDebug in class weka.classifiers.Classifier</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>d - a boolean that indicates whether debugging should be on or off</td>
</tr>
<tr>
<td><code>public long getSeed()</code></td>
<td>Returns: the current seed for the random number generator</td>
</tr>
<tr>
<td><code>public void setSeed(long seed)</code></td>
<td>Parameters: seed - set a new seed for the random number generator</td>
</tr>
<tr>
<td><code>public double classifyInstance(weka.core.Instance inst)</code></td>
<td>Throws java.lang.Exception</td>
</tr>
<tr>
<td><strong>Overrides:</strong></td>
<td>classifyInstance in class weka.classifiers.Classifier</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>inst - the instance to classify</td>
</tr>
<tr>
<td><strong>Returns:</strong></td>
<td>the class that is assigned to this instance or -1x</td>
</tr>
<tr>
<td><strong>Throws:</strong></td>
<td>java.lang.Exception</td>
</tr>
<tr>
<td><code>public void buildClassifier(weka.core.Instances data)</code></td>
<td>Throws java.lang.Exception</td>
</tr>
<tr>
<td><strong>Specified by:</strong></td>
<td>buildClassifier in class weka.classifiers.Classifier</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
<td>data - the training set</td>
</tr>
<tr>
<td><strong>Throws:</strong></td>
<td>java.lang.Exception</td>
</tr>
</tbody>
</table>
toOptionString
public java.lang.String toOptionString()

Returns:
a string representation of the Learner, including all parameters (if any). If there are parameters, the
returned string will start and end with quotes. Thus the representation is suitable for the command-
line (e.g., for initializing other objects).

toString
public java.lang.String toString()

Overrides:
toString in class java.lang.Object

Returns:
a string representation of the learner. Currently it returns the string representation of the current rule.

listOptions
public java.util.Enumeration listOptions()

Returns an enumeration describing the available options. Valid options are:
-D turn on the debug mode and produce a lot of output -H heuristic
the heuristic used for evaluating candidate rules. If the heuristic takes parameters, these can be added
as a string (e.g., -H "seco.heuristics.MEstimate -M 2") (Default: seco.heuristics.Laplace) -S
the seed for the random number generator (Default: 1) -C
the index of the class for which the rule should be learned (Default: 0) *

Specified by:
listOptions in interface weka.core.OptionHandler

Overrides:
listOptions in class weka.classifiers.Classifier

Returns:
an enumeration of all available options

setOptions
public void setOptions(java.lang.String[] options)

throws java.lang.Exception
parse a list of options.

Specified by:
setOptions in interface weka.core.OptionHandler

Overrides:
setOptions in class weka.classifiers.Classifier

Parameters:
opts - the list of options an array of strings

Throws:
Exception - if an option is not supported
java.lang.Exception

getOptions
public java.lang.String[] getOptions()

the current settings of the classifier

Specified by:
getOptions in interface weka.core.OptionHandler

Overrides:
getOptions in class weka.classifiers.Classifier

Returns:
an array of strings suitable for passing to setOptions

enumerateMeasures
public java.util.Enumeration enumerateMeasures()
Returns an enumeration of the additional measure names

**Specified by:**
enumerateMeasures in interface weka.core.AdditionalMeasureProducer

**Returns:**
an enumeration of the measure names

---

getMeasure
public double getMeasure(java.lang.String additionalMeasureName)
Returns the value of the named measure

**Specified by:**
getMeasure in interface weka.core.AdditionalMeasureProducer

**Parameters:**
measureName - the name of the measure to query for its value

**Returns:**
the value of the named measure

**Throws:**
java.lang.IllegalArgumentException - if the named measure is not supported

---

main
public static void main(java.lang.String[] args)
Main method.

---

Class / Interface 32: TopDownBeamSearch

public class TopDownBeamSearch
extends weka.classifiers.Classifier
implements weka.core.WeightedInstancesHandler, weka.core.AdditionalMeasureProducer,
weka.core.OptionHandler

The seco package implements generic functionality for simple separate-and-conquer rule learning.
TopDownBeamSearch implements a top-down (general-to-specific) induction of a single rule via
beam search. In principle, a Rule can be used as a classifier in its own right, but the main purpose of
this is to provide a learner for the inner loop of the separate-and-conquer/covering loop. TODO: -
Stopping heuristics - Filtering heuristics (at the place where we are now " ---> ignored") - maybe
initialization with a given rule (be careful not to re-initialize the rule, init used vector) - local
optimization (gain heuristics do not allow a global comparison between rules, such as Foil-Gain) -
LEFs (multiple evaluation heuristics) - cleaner interface should maybe store a Rule (and not a
CandidateRule) internally and copy it to a CandidateRule for learning - split value can be chosen
from domain or interpolate between neighboring values - Maybe the search should maintain a general
confusion matrix so that CN2 can implemented (both, the entropy heuristic and the decision list
search that auto-picks the class) - have I covered missing class values? - the two code blocks for
finding the best condition should probably be separate routines/methods of some class. - is it possible
to order to conditions so that certain combinations do not need to be re-searched? I guess that works
only for exhaustive OPUS... However, there might be duplicate rules in the beam now! Parts of it is
based on code for JRip and for Prism.

**Constructor Detail**

TopDownBeamSearch
public TopDownBeamSearch()

---

**Method Detail**
**initRule**

```java
public void initRule()
initialize the rule learner with a fresh, empty rule.
```

**Parameters:**
- `initRule`: initialize the rule learner with

**Rule:**
- `initRule(CandidateRule r)`

**getRule**

```java
public CandidateRule getRule()
Returns:
the learned rule
```

**getHeuristic**

```java
public SearchHeuristic getHeuristic()
Returns:
a `seco.heuristics.SearchHeuristic`
```

**setHeuristic**

```java
public void setHeuristic(SearchHeuristic h)
set the heuristic used for evaluating rules
```

**Parameters:**
- `h`: a `seco.heuristics.SearchHeuristic`

**getBeamWidth**

```java
public long getBeamWidth()
Returns:
the beam width (the number of candidate rules remembered for learning)
```

**setBeamWidth**

```java
public void setBeamWidth(int beam)
Parameters:
beam - set the beam width
```

**getTargetClass**

```java
public double getTargetClass()
Returns:
the index of the class for which a rule is learned
```

**setTargetClass**

```java
public void setTargetClass(double i)
set the index of the class to a new value. it also resets the rule (initRule).
```

**debug**

```java
public boolean debug()
Returns:
true if debugging is on, false if not
```

**setDebug**

```java
public void setDebug(boolean d)
Overrides:
setDebug in class weka.classifiers.Classifier
```
### Parameters:
- **d**: a boolean that indicates whether debugging should be on or off

#### getSeed
```
public long getSeed()
```
**Returns:**
the current seed for the random number generator

#### setSeed
```
public void setSeed(long seed)
```
**Parameters:**
- **seed**: set a new seed for the random number generator

#### classifyInstance
```
public double classifyInstance(weka.core.Instance inst)
```
**Throws:**
java.lang.Exception
**Overrides:**
classifyInstance in class weka.classifiers.Classifier
**Parameters:**
- **inst**: the instance to classify
**Returns:**
the class that is assigned to this instance or -1x

#### buildClassifier
```
public void buildClassifier(weka.core.Instances data)
```
**Specified by:**
buildClassifier in class weka.classifiers.Classifier
**Parameters:**
- **data**: the training set
**Throws:**
java.lang.Exception

#### toOptionString
```
public java.lang.String toOptionString()
```
**Returns:**
a string representation of the Learner, including all parameters (if any). If there are parameters, the returned string will start and end with quotes. Thus the representation is suitable for the command-line (e.g., for initializing other objects).

#### toString
```
public java.lang.String toString()
```
**Overrides:**
toString in class java.lang.Object
**Returns:**
a string representation of the learner. Currently it returns the string representation of the current rule.

#### listOptions
```
public java.util.Enumeration listOptions()
```
**Returns**
an enumeration describing the available options. Valid options are:
-D

turn on the debug mode and produce a lot of output -H heuristic

the heuristic used for evaluating candidate rules. If the heuristic takes parameters, these can be added as a string (e.g., -H "seco.heuristics.MEstimate -M 2") (Default: seco.heuristics.Laplace) -B beam-width

the beam width used for searching. This specifies the number of candidates that will be remembered after each refinement. If the value is inf (infinity), an exhaustive best-first search is performed. A beam width of 1 results in hill-climbing or gradient ascent search. (Default: 1) -S

the seed for the random number generator (Default: 1) Not yet valid are: -C

the index of the class for which the rule should be learned (Default: 0) *

Specified by:
listOptions in interface weka.core.OptionHandler

Overrides:
listOptions in class weka.classifiers.Classifier

Returns:
an enumeration of all available options

---

setOptions

```java
public void setOptions(String[] options)
throws java.lang.Exception
```

take a list of options.

Specified by:
setOptions in interface weka.core.OptionHandler

Overrides:
setOptions in class weka.classifiers.Classifier

Parameters:
opts - the list of options an array of strings

Throws:
Exception - if an option is not supported
java.lang.Exception

---

getOptions

```java
public java.lang.String[] getOptions()
```

take the current settings of the classifier

Specified by:
getOptions in interface weka.core.OptionHandler

Overrides:
getOptions in class weka.classifiers.Classifier

Returns:
an array of strings suitable for passing to setOptions

---

enumerateMeasures

```java
public java.util.Enumeration enumerateMeasures()
```

Returns an enumeration of the additional measure names

Specified by:
enumerateMeasures in interface weka.core.AdditionalMeasureProducer

Returns:
an enumeration of the measure names

---

getMeasure

```java
public double getMeasure(String additionalMeasureName)
```

Returns the value of the named measure

Specified by:
getMeasure in interface weka.core.AdditionalMeasureProducer

Parameters:
<table>
<thead>
<tr>
<th>measureName - the name of the measure to query for its value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Returns:</strong> the value of the named measure</td>
</tr>
<tr>
<td><strong>Throws:</strong> java.lang.IllegalArgumentException - if the named measure is not supported</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>main</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static void <strong>main</strong>(java.lang.String[] args)</td>
</tr>
<tr>
<td>Main method.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>getPruner</th>
</tr>
</thead>
<tbody>
<tr>
<td>public PruningTemplate <strong>getPruner</strong>()</td>
</tr>
<tr>
<td><strong>Returns:</strong> the m_prune</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>setPruner</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void <strong>setPruner</strong>(PruningTemplate m_prune)</td>
</tr>
<tr>
<td><strong>Parameters:</strong></td>
</tr>
<tr>
<td>m_prune - the m_prune to set</td>
</tr>
</tbody>
</table>
C4 – Paket „seco.models“

<table>
<thead>
<tr>
<th>Klassen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>CandidateRule</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>Condition</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>NominalCondition</td>
<td>Conditions with nominal values</td>
</tr>
<tr>
<td>NumericCondition</td>
<td>Conditions with numeric values</td>
</tr>
<tr>
<td>Rule</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>RuleSet</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>TrueCondition</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
<tr>
<td>ValueTestCondition</td>
<td>The seco package implements generic functionality for simple separate-and-conquer rule learning.</td>
</tr>
</tbody>
</table>

Anhang C - 5: Paketinhalt des Pakets "seco.models"

Class / Interface 33: CandidateRule

public class CandidateRule
extends Rule
The seco package implements generic functionality for simple separate-and-conquer rule learning on top of weka. CandidateRule is a subclass of Rule for candidate rule. A candidate rule contains additional information like - a slot for storing the result of the rule - the history of the rule (a pointer back to the predecessor) Parts of it is based on code for JRip and for Prism.

Constructor Detail

CandidateRule
public CandidateRule()

CandidateRule
public CandidateRule(Condition head)

Method Detail

copy
public java.lang.Object copy()

Specified by:
copy in interface weka.core.Copyable

Overrides:
copy in class Rule

Returns:
a shallow copy of the candidate rule The copy does not copy the conditions, while the clone does.

clone
public java.lang.Object clone()

throws java.lang.CloneNotSupportedException

Overrides:
clone in class Rule

Returns:
a deep copy of the candidate rule the clone contains also contains fresh copies of the conditions and the coverage stats.

**Throws:**
java.lang.CloneNotSupportedException

<table>
<thead>
<tr>
<th>shadow</th>
</tr>
</thead>
<tbody>
<tr>
<td>public java.lang.Object <strong>shadow</strong>()</td>
</tr>
<tr>
<td>throws java.lang.CloneNotSupportedException</td>
</tr>
</tbody>
</table>

**Overrides:**
shadow in class Rule

<table>
<thead>
<tr>
<th>Returns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a copy of the rule, but with empty body and empty stats.</td>
</tr>
</tbody>
</table>

**Throws:**
java.lang.CloneNotSupportedException

<table>
<thead>
<tr>
<th>initBody</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void <strong>initBody</strong>()</td>
</tr>
<tr>
<td>reset the body of the rule to an empty vector and reset all statistics to 0.</td>
</tr>
</tbody>
</table>

**Overrides:**
initBody in class Rule

<table>
<thead>
<tr>
<th>betterRule</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static CandidateRule <strong>betterRule</strong>(CandidateRule r1, CandidateRule r2)</td>
</tr>
<tr>
<td>return the better of two rules. It is assumed that computeRuleValue has been previously called! A rule is better if its evaluation is higher or if the evaluation is the same, but it has shorter length, or the same length and a higher random tie break value. If one of the two is null, the other is returned.</td>
</tr>
</tbody>
</table>

**Parameters:**
- r1 - the first candidate rule
- r2 - the second candidate rule

<table>
<thead>
<tr>
<th>Returns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>the better of the two rules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>compareTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int <strong>compareTo</strong>(java.lang.Object o)</td>
</tr>
<tr>
<td>throws java.lang.ClassCastException</td>
</tr>
<tr>
<td>compare a rule to another object. If the object is another CandidateRule this function behaves like compareTo(CandidateRule). Otherwise, it throws a ClassCastException (as CandidateRules are comparable only to other CandidateRules).</td>
</tr>
</tbody>
</table>

**Specified by:**
compareTo in interface java.lang.Comparable

**Parameters:**
- o - the object to compare to

<table>
<thead>
<tr>
<th>Returns:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 if the o is better, 1 if the old rule is better, 0 else</td>
</tr>
</tbody>
</table>

| Throws: |
| java.lang.ClassCastException - if the object is not a rule |

<table>
<thead>
<tr>
<th>compareTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>public int <strong>compareTo</strong>(CandidateRule r)</td>
</tr>
<tr>
<td>throws java.lang.NullPointerException</td>
</tr>
<tr>
<td>compare two rules. It is assumed that computeRuleValue has been previously called! A rule is better if its evaluation is higher or if the evaluation is the same, but it has shorter length, or the same length and a higher random tie break value.</td>
</tr>
</tbody>
</table>

**Parameters:**

o - the rule to compare to

**Returns:**
-1 if the o is better, 1 if the old rule is better, 0 else

**Throws:**
java.lang.NullPointerException - if the object is null

equals
public boolean equals(CandidateRule r)
compare two rules, if the class and all conditions are equal the two rules are considered to be the same.

**Parameters:**
r - the rule to test the equality with

**Returns:**
true if the rules are equal

```

equals
```

calculateRuleValue
public double calculateRuleValue(SearchHeuristic h)
compute the heuristic evaluation of the rule with the provided rule value. This calls passes the object itself to the provided heuristic and internally stores the result.

**Parameters:**
h - a search heuristic

**Returns:**
the computed rule value

```
calculateRuleValue
```

generateRuleValue
public double generateRuleValue()
generate a rule value that has been previously computed.

```
generateRuleValue
```

generateTieBreaker
public double generateTieBreaker()
generate a tie break value, i.e., a random number between 0 and 1. This number is always the same unless it is reset in the mean-time.

```
generateTieBreaker
```

generateTieBreaker
public void generateTieBreaker()
reset the tie break value. This causes the next call to generateTieBreaker to return a new value.
generateTieBreaker should be called whenever the rule candidate changes (e.g., by adding or deleting conditions). This is *not* done automatically. resetTieBreaker is automatically called only when copying or constructing a rule candidate.

```
generateTieBreaker
```

```
setRandom(java.util.Random r)
```
set the internal random generator to an initialized Random object.

```
setRandom
```

generatePredecessor
public CandidateRule generatePredecessor()
generate the predecessor of the candidate rule

**Returns:**
the predecessor or null

```
generatePredecessor
```

```
setPredecessor(CandidateRule r)
```
specialize
public CandidateRule specialize(Condition c)
return a specialization of the current rule. The specialization will be a fresh copy and the current rule
will be its predecessor.

**Parameters:**
c - a condition

**Returns:**
a specialization of the rule that results from adding c

toString
public java.lang.String toString()
print out a candidate rule with coverage statistics and the heuristic value` @return a printable
representation of the rule

**Overrides:**
toString in class Rule

---

**Class / Interface 34: Condition**

public abstract class Condition
extends java.lang.Object
implements weka.core.Copyable
The seco package implements generic functionality for simple separate-and-conquer rule learning.
Condition implements conditions for rules Parts of it is based on code for JRip and for Prism.

**Constructor Detail**

Condition
public Condition()

Condition
public Condition(weka.core.Attribute a)

Condition
public Condition(weka.core.Attribute a,
    double value)

Condition
public Condition(weka.core.Attribute a,
    double value,
    boolean cmp)

**Method Detail**

getAttr
public weka.core.Attribute getAttr()

getValue
public double getValue()

setValue
public void setValue(double v)
### Class / Interface 35: NominalCondition

```java
public class NominalCondition
    extends Condition
    implements java.lang.Cloneable, java.io.Serializable
Conditions with nominal values
```

#### Constructor Detail

NominalCondition
public **NominalCondition**(weka.core.Attribute a)

NominalCondition
public **NominalCondition**(weka.core.Attribute a,
                           double value)

NominalCondition
public **NominalCondition**(weka.core.Attribute a,
                           double value,
                           boolean cmp)
```
Method Detail

covers
public boolean covers(weka.core.Instance inst)
check whether the instance is covered by this condition

Specified by:
covers in class Condition

Parameters:
inst - the instance in question

Returns:
the boolean value indicating whether the instance is covered by this antecedent

toString
public java.lang.String toString()

Specified by:
toString in class Condition

Class / Interface 36: NumericCondition

class NumericCondition
extends Condition
implements java.lang.Cloneable, java.io.Serializable

Conditions with numeric values

Constructor Detail

NumericCondition
public NumericCondition(weka.core.Attribute a)

NumericCondition
public NumericCondition(weka.core.Attribute a, double value)

NumericCondition
public NumericCondition(weka.core.Attribute a, double value, boolean cmp)

Method Detail
covers
public boolean covers(weka.core.Instance inst)
check Whether the instance is covered by this condition

Specified by:
covers in class Condition

Parameters:
inst - the instance in question

Returns:
the boolean value indicating whether the instance is covered by this antecedent

toString
public java.lang.String toString()
Specified by:
toString in class Condition

<table>
<thead>
<tr>
<th>Class / Interface 37: Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>public class Rule</td>
</tr>
<tr>
<td>extends java.lang.Object</td>
</tr>
<tr>
<td>implements weka.core.Copyable, java.lang.Cloneable, java.io.Serializable</td>
</tr>
</tbody>
</table>

The seco package implements generic functionality for simple separate-and-conquer rule learning on top of weka. Rule implements the representation of a single rule. Note that in principle, a Rule can be used as a classifier in its own right. Parts of it is based on code for JRip and for Prism.

Constructor Detail

Rule
public Rule()

Rule
public Rule(Condition head)

Method Detail

getMaxLen
public int getMaxLen()

setMaxLen
public void setMaxLen(int maxLen)

copy
public java.lang.Object copy()

Specified by:
copy in interface weka.core.Copyable

Returns:
a shallow copy of the list. the copy is shallow in the sense that the conditions are not copied (i.e., both the original and the copy point to the same conditions). However, all statistics are copied, so that both rules can be evaluated and used independently.

cache
public java.lang.Object clone()

Overrides:
cache in class java.lang.Object

Returns:
a deep copy of the list. the clone contains also contains fresh copies of the conditions.

Throws:
java.lang.CloneNotSupportedException

shadow
public java.lang.Object shadow()

Throws:
java.lang.CloneNotSupportedException

Returns:
a copy of the rule, but with empty body and empty stats.
### java.lang.CloneNotSupportedException

**getHead**

```java
public Condition getHead()
```

**Returns:**
the head of the rule, a condition on the class attribute

**setHead**

```java
public void setHead(Condition h)
```

set the head of the rule to a new class condition

**getPredictedValue**

```java
public double getPredictedValue()
```

**Returns:**
the class value predicted by the rule

**getBody**

```java
public weka.core.FastVector getBody()
```

**Returns:**
the body of the rule, a FastVector of Conditions

**initBody**

```java
public void initBody()
```

reset the body of the rule to an empty vector and the stats to 0

**getLastCondition**

```java
public Condition getLastCondition()
```

**Returns:**
the final condition of the body.

**getCondition**

```java
public Condition getCondition(int n)
```

**Parameters:**
n - the number of the condition that should be returned

**Returns:**
the nth condition of the body.

**addCondition**

```java
public void addCondition(Condition c)
```

add a condition to the body of the rule

**deleteLastCondition**

```java
public void deleteLastCondition()
```

delete the last condition from the body of the rule

**deleteCondition**

```java
public void deleteCondition(int n)
```

delete the nth condition from the body of the rule

**Parameters:**
n - number of the condition to delete

**replaceLastCondition**

```java
public void replaceLastCondition(Condition c)
```

replace the last condition from the body of the rule with a new condition

**Parameters:**
c - new condition

replaceCondition
public void replaceCondition(Condition c, int n)
replace the nth condition from the body of the rule with a new condition

Parameters:
c - new condition
n - number of condition to replace

getStats
public TwoClassStats getStats()
return the TwoClassStats object containing the coverage counts

setStats
public void setStats(TwoClassStats stats)
set the TwoClassStats object to a new object

coveredInstances
public weka.core.Instances coveredInstances(weka.core.Instances data)
return the set of Instances that are covered by the rule.

Parameters:
data - the set of instances

Returns:
the set of instances that are not covered.

uncoveredInstances
public weka.core.Instances uncoveredInstances(weka.core.Instances data)
return the set of Instances that are not covered by the rule.

Parameters:
data - the set of instances

Returns:
the set of instances that are not covered.

length
public int length()
get the length of the rule, the number of conditions in the body

classifyInstance
public double classifyInstance(weka.core.Instance inst)
classify the passed Instance, i.e. return the class value if the instance is covered by the rule, or the missing value if it is not covered by the rule.

Parameters:
inst - the instance

Returns:
the class of the instance or a missing value

covers
public boolean covers(weka.core.Instance inst)
check whether the rule covers an instance

Parameters:
inst - the instance to check

Returns:
true if the rule covers the instance, false else
public class RuleSet
extends java.lang.Object
implements java.io.Serializable
The seco package implements generic functionality for simple separate-and-conquer rule learning. RuleSet implements the representation of a rule set. TODO: - Should the default rule be stored as another rule?

Class / Interface 38: RuleSet

Scope: public
type: java.lang

Public Class

Class / Interface 38: RuleSet

Class / Interface 38: RuleSet

Class / Interface 38: RuleSet

Class / Interface 38: RuleSet

Class / Interface 38: RuleSet

Class / Interface 38: RuleSet

Method Detail

setRuleSetStats

Scope: public
type: void

Method

setRuleSetStats

Scope: public
type: RuleSetStats

Parameter

setRuleSetStats(RuleSetStats s)

getStats

Scope: public
type: RuleSetStats

Method

getStats()

getRules

Scope: public
type: weka.core.FastVector

Method

getRules()

Returns:
the rule set, a FastVector of Rules. Note that the default rule is not returned.

getRule

Scope: public
type: Rule

Method

getRule(int n)

Parameters:

n - the number of the rule should be returned

Returns:
the nth rule

numRules

Scope: public
type: int

Method

numRules()

Returns:
the number of rules (excluding the default rule)

numConditions

Scope: public
type: int

Method

numConditions()

Returns:
the number of conditions in the rules

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addRule(Rule c)</code></td>
<td>add a rule to the set</td>
<td><code>c</code> - the rule to be added</td>
</tr>
<tr>
<td><code>deleteLastRule()</code></td>
<td>delete the last rule</td>
<td></td>
</tr>
<tr>
<td><code>deleteRule(int n)</code></td>
<td>delete the nth rule</td>
<td><code>n</code> - number of the rule to delete</td>
</tr>
<tr>
<td><code>replaceLastRule(Rule r)</code></td>
<td>replace the last rule with a new rule</td>
<td><code>r</code> - new rule</td>
</tr>
<tr>
<td><code>replaceCondition(Rule r, int n)</code></td>
<td>replace the nth rule with a new rule</td>
<td><code>r</code> - new rule, <code>n</code> - number of rule to replace</td>
</tr>
<tr>
<td><code>getDefaultPrediction()</code></td>
<td>get the default prediction</td>
<td></td>
</tr>
<tr>
<td><code>getDefaultRule()</code></td>
<td>get the default rule</td>
<td></td>
</tr>
<tr>
<td><code>setDefaultRule(Rule r)</code></td>
<td>set the default rule to a new rule</td>
<td><code>r</code> - the new rule</td>
</tr>
<tr>
<td><code>getCoveringRules(weka.core.Instance inst)</code></td>
<td>return all rules that cover a given instance</td>
<td><code>inst</code> - the instance</td>
</tr>
<tr>
<td><code>getFirstCoveringRule</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public Rule getFirstCoveringRule(weka.core.Instance inst)
return the first rule that covers a given instance or null if no instance covers the instance. The default rule is not tested.

**Parameters:**
inst - the instance

**Returns:**
a the first rule that cover the instance

classifyInstance
public double classifyInstance(weka.core.Instance inst)
throws java.lang.Exception
classify the passed Instance with the rule set. Currently we assume classification as a decision list, i.e., the prediction of the first rule that doesn't predict the missing value is returned. Eventually, this should probably be a parameter or maybe even a separate subclass.

**Parameters:**
inst - the instance

**Returns:**
the class of the instance or a missing value

**Throws:**
java.lang.Exception

toString
public java.lang.String toString()

**Overrides:**
toString in class java.lang.Object

**Returns:**
a printable version of a rule set.

---

**Class / Interface 39: TrueCondition**
public abstract class TrueCondition extends Condition
implements weka.core.Copyable
The seco package implements generic functionality for simple separate-and-conquer rule learning. Condition implements conditions for rules. True is a condition that is always true.

**Constructor Detail**

TrueCondition
public TrueCondition()

**Method Detail**

coveredInstances
public weka.core.Instances coveredInstances(weka.core.Instances data)
return the list of instances that are covered by the condition

**Overrides:**
coveredInstances in class Condition

**Parameters:**
data - the list of instances

**Returns:**
a new list of covered instances
```java
covers
public boolean covers(weka.core.Instance inst)
Specified by:
covers in class Condition

toString
public java.lang.String toString()
Specified by:
toString in class Condition
```

### Class / Interface 40: ValueTestCondition

public abstract class ValueTestCondition extends Condition
implements weka.core.Copyable

The seco package implements generic functionality for simple separate-and-conquer rule learning. Condition implements conditions for rules. ValueTestCondition implements conditions that test for certain value of attributes. They may be Nominal conditions or Numeric Conditions. Parts of it is based on code for JRip and for Prism.

#### Constructor Detail

**ValueTestCondition**

public ValueTestCondition(weka.core.Attribute a)

**ValueTestCondition**

public ValueTestCondition(weka.core.Attribute a, double value)

**ValueTestCondition**

public ValueTestCondition(weka.core.Attribute a, double value, boolean cmp)

#### Method Detail

**getAttr**

public weka.core.Attribute getAttr()

Overrides:
getAttr in class Condition

**getValue**

public double getValue()

Overrides:
getValue in class Condition

**setValue**

public void setValue(double v)

Overrides:
setValue in class Condition

```
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean <code>cmp()</code></td>
<td><code>Overrides: cmp in class Condition</code></td>
</tr>
<tr>
<td>public void <code>setCmp</code></td>
<td><code>Overrides: setCmp in class Condition</code></td>
</tr>
</tbody>
</table>
C5 – Paket „seco.pruning“

<table>
<thead>
<tr>
<th>Klassen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>IREPPruning</td>
<td>Implements the IREP pruning behavior.</td>
</tr>
<tr>
<td>NoPruning</td>
<td>Dummyclass if no Pruning is specified.</td>
</tr>
<tr>
<td>Prepruning</td>
<td>Class implementing Prepruning behavior.</td>
</tr>
<tr>
<td>REPPruning</td>
<td>Implements the REP Pruningalgorithm.</td>
</tr>
<tr>
<td>TDPPruning</td>
<td>This Class implements the TDP-Pruning Algorithm</td>
</tr>
</tbody>
</table>

Class / Interface 41: IREPOpt

class IREPOpt
extends PruningTemplate


Valid options are:
- D debug flag
Flag wether debug mode is enabled or not.

Constructor Detail

IREPOpt
public IREPOpt()

Method Detail

outerSplit
public weka.core.Instances[] outerSplit(weka.core.Instances data)
throws java.lang.Exception

Description copied from class: PruningTemplate
Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size, and the pruning set (Instances[1]), which contains 1/3 of the initial set size.
The growing set is used for learning and the pruning set is used for pruning

Specified by:
outerSplit in class PruningTemplate
Parameters:
- data - the initial set of examples
Returns:
- instances[0]: the growing set;
- instances[1]: the pruning set

Throws:
- java.lang.Exception

innerSplit
public weka.core.Instances[] innerSplit(weka.core.Instances data)
throws java.lang.Exception

Description copied from class: PruningTemplate
the trainingexamples in the covering -loop Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size, and the pruning set (Instances[1]), which contains 1/3 of the initial set size.
The growing set is used for learning and the pruning set is used for pruning

**Specified by:**
innerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

clauseStop

```java
public boolean clauseStop(RuleSet rs,
    CandidateRule r,
    weka.core.Instances data)
throws java.lang.Exception
```

**Description copied from class: PruningTemplate**
A generic method. Implements the ClauseStoppingCriterion. Must be overwritten for each new Pruning algorithm or at least return the Object which is passed to it.

**Specified by:**
clauseStop in class PruningTemplate

**Parameters:**
rs - The RuleSet to prune.
data - The test set

**Returns:**
Returns Whether to stop rule learning (true) or not (false)

**Throws:**
java.lang.Exception

literalStop

```java
public boolean literalStop(CandidateRule r,
    weka.core.Instances data)
throws java.lang.Exception
```

**Implements the LiteralStoppingCriterion. Is always false**

**Specified by:**
literalStop in class PruningTemplate

**Parameters:**
r - The CandidateRule
data - The test set

**Returns:**
Returns Whether to stop rule refining (true) or not (false)

**Throws:**
java.lang.Exception

postProcess

```java
public RuleSet postProcess(RuleSet rs,
    weka.core.Instances data)
throws java.lang.Exception
```

**Optimization Stage like the RIPPER's**

**Specified by:**
postProcess in class PruningTemplate

**Parameters:**
rs - The ruleset to be optimized
data - The data for optimization

**Returns:**
The optimized ruleset

**Throws:**
java.lang.Exception

---

```java
postProcessRule
public CandidateRule postProcessRule(CandidateRule r,
weka.core.Instances data)
    throws java.lang.Exception
Prunes a rule on the given data.
**Specified by:**
postProcessRule in class PruningTemplate
**Parameters:**
r - The rule to prune
data - The pruning set
**Returns:**
the pruned rule or r, iff no better rule is found
**Throws:**
java.lang.Exception
```

---

```java
setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
**Description copied from class: PruningTemplate**
Sets the options for the Pruning algorithms.
**Specified by:**
setOptions in interface weka.core.OptionHandler
**Overrides:**
setOptions in class PruningTemplate
**Parameters:**
options - - The options
**Throws:**
java.lang.Exception
```

---

### Class / Interface 42: IREPruning

public class IREPruning
extends PruningTemplate
Implements the REP Pruning algorithm. Valid options are:
- **-D** debug flag
  Flag wether debug mode is enabled or not.
- **-P** pruning flag
  Flag wether pruning is enabled or not.
- **-N** number
  Set number of folds for pruning. One fold is used as the pruning set. (Default: 3)
- **-SH** Set the heuristic for Pruning (Default: seco.heuristics.Laplace)
  -LC Literalstopping Criterion
  Set the Literalstopping Criterion. (Default: seco.pruning.criterion.NoStopping)
- **-CC** Clausestopping Criterion
  Set the Clausestopping Criterion. (Default: seco.pruning.criterion.NoStopping)
- **-OP**
  Specifies the RuleOperator for Pruning (Default: seco.pruning.operator.RuleDeleteLastCondition).
  TODO:
  - Specification of pruning operators. Use 1,2,...or all operators.
Constructor Detail

IREPruning
public IREPruning()

Method Detail

postProcessRule
public CandidateRule postProcessRule(CandidateRule r,
            weka.core.Instances data)
    throws java.lang.Exception
Prunes a Rule.

1: curBest = r
2: r' = r
3: do
4:   r' = apply operator p to r'
5:   if Accuracy(r') < Accuracy(curBest) then
6:     break;
7:   curBest = r';
8: while( a better rule can be found )

Specified by:
postProcessRule in class PruningTemplate

Parameters:
r - - The CandidateRule to prune.
data - - The pruning set
Returns:
Returns a pruned CandidateRule
Throws:
java.lang.Exception

postProcess
public RuleSet postProcess(RuleSet rs,
            weka.core.Instances data)
    throws java.lang.Exception
Does nothing

Specified by:
postProcess in class PruningTemplate

Parameters:
rs - - The ruleset to prune
data - - The pruning set
Returns:
The best simplification for a ruleset
Throws:
java.lang.Exception

clauseStop
public boolean clauseStop(RuleSet rs,
            CandidateRule r,
            weka.core.Instances data)
    throws java.lang.Exception
Implements the ClauseStoppingCriterion. Defers the decision to the LiteralStoppingCriterion,
because the decision for the RuleStop is only based on one Rule

**Specified by:**
clauseStop in class PruningTemplate

**Parameters:**
rs - The RuleSet
r - The CandidateRule
data - The test set

**Returns:**
Whether to stop rule learning (true) or not (false)

**Throws:**
java.lang.Exception

---

**literalStop**

public boolean **literalStop**(CandidateRule r, weka.core.Instances data)

Implements the LiteralStoppingCriterion. Is always false

**Specified by:**
literalStop in class PruningTemplate

**Parameters:**
r - The CandidateRule
data - The test set

**Returns:**
Returns Whether to stop rule refining (true) or not (false)

**Throws:**
java.lang.Exception

---

**setOptions**

public void **setOptions**(java.lang.String[] options)
throws java.lang.Exception

**Description copied from class:** PruningTemplate

Sets the options for the Pruningalgorithms.

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Overrides:**
setOptions in class PruningTemplate

**Parameters:**
options - - The options

**Throws:**
java.lang.Exception

---

**listOptions**

public java.util.Enumeration **listOptions**()

**Specified by:**
listOptions in interface weka.core.OptionHandler

**Overrides:**
listOptions in class PruningTemplate

---

**outerSplit**

public weka.core.Instances[] **outerSplit**(weka.core.Instances data)
throws java.lang.Exception

Splits the examples into to set.
The growing set (Instances[0]), which contains (num_folds-1)/num_folds of the initial instance set size,
and the pruning set (Instances[1]), which contains 1/num_folds of the initial set size.
The growing set is used for learning and the pruning set is used for pruning. If num_folds is smaller
than zero then the set is not split.

**Specified by:**
outerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

```java
public weka.core.Instances[] innerSplit(weka.core.Instances data)
    throws java.lang.Exception
```

Splits the examples into two sets.
The growing set (Instances[0]), which contains \((\text{num}\_\text{folds}-1)/\text{num}\_\text{folds}\) of the initial instance set size,
and the pruning set (Instances[1]), which contains \(1/\text{num}\_\text{folds}\) of the initial set size.
The growing set is used for learning and the pruning set is used for pruning. If num_folds is smaller than zero then the set is not split.

**Specified by:**
innerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

**Class / Interface 43: NoPruning**

public class NoPruning
extends PruningTemplate

Dummy class if no Pruning is specified. All Methods are dummy Methods which do nothing or return a default value

**Constructor Detail**

NoPruning
public NoPruning()

**Method Detail**

postProcess
public java.lang.Object postProcess(java.lang.Object o,
    weka.core.Instances data)
    throws java.lang.Exception

Does nothing

**Throws:**
java.lang.Exception

---
public boolean clauseStop(RuleSet rs, 
        CandidateRule r, 
        weka.core.Instances data)
A generic method. Implements the ClauseStoppingCriterion. Is always false.

Specified by:
clauseStop in class PruningTemplate

Parameters:
rs - The RuleSet
r - The CandidateRule
data - The test set

Returns:
false

Throws:
java.lang.Exception

public boolean literalStop(CandidateRule r, 
        weka.core.Instances data)
A generic method. Implements the LiteralStoppingCriterion. Is always false.

Specified by:
literalStop in class PruningTemplate

Parameters:
r - The CandidateRule
data - The test set

Returns:
false

Throws:
java.lang.Exception

public RuleSet postProcess(RuleSet rs, 
        weka.core.Instances data)
throws java.lang.Exception
A generic method for pruning.

Specified by:
postProcess in class PruningTemplate

Parameters:
rs - The RuleSet to prune.
data - The pruningset

Returns:
Returns rs

Throws:
java.lang.Exception

public CandidateRule postProcessRule(CandidateRule r, 
        weka.core.Instances data)
throws java.lang.Exception
A generic method for pruning.

Specified by:
postProcessRule in class PruningTemplate

Parameters:
r - The CandidateRule to prune.
data - The pruningset

Returns:
innerSplit
public weka.core.Instances[] innerSplit(weka.core.Instances data)
        throws java.lang.Exception

    Does nothing

    Specified by:
    innerSplit in class PruningTemplate

    Parameters:
    data - the initial set of examples

    Returns:
    the initial set of examples

    Throws:
    java.lang.Exception

outerSplit
public weka.core.Instances[] outerSplit(weka.core.Instances data)
        throws java.lang.Exception

    Does nothing

    Specified by:
    outerSplit in class PruningTemplate

    Parameters:
    data - the initial set of examples

    Returns:
    the initial set of examples

    Throws:
    java.lang.Exception

Class / Interface 44: PrePruning
public class PrePruning
extends PruningTemplate
Class implementing Prepruning behavior. The Examples arent split and only the Methods for the
Stoppingcriteria are overwritten.

Constructor Detail

PrePruning
public PrePruning()

Method Detail

clauseStop
public boolean clauseStop(RuleSet rs,
        CandidateRule r,
        weka.core.Instances data)

    Implements the ClauseStoppingCriterion. Is always false.

    Specified by:
    clauseStop in class PruningTemplate

    Parameters:
    rs - The RuleSet
public boolean literalStop(CandidateRule r, weka.core.Instances data)
    throws java.lang.Exception

Implements the LiteralStoppingCriterion. Delegates the decision to the StoppingCriterion

Specified by:
literalStop in class PruningTemplate

Parameters:
r - The CandidateRule
data - The test set

Returns: Returns Whether to stop rule refining (true) or not (false)

Throws: java.lang.Exception

public RuleSet postProcess(RuleSet rs, weka.core.Instances data)
    throws java.lang.Exception

A generic method for pruning. Must be overwritten for each new Pruningalgorithm or at least return
the Object which is passed to it.

Specified by:
postProcess in class PruningTemplate

Parameters:
rs - - The RuleSet to prune.
data - - The pruningset

Returns: Returns a pruned RuleSet

Throws: java.lang.Exception

public CandidateRule postProcessRule(CandidateRule r, weka.core.Instances data)
    throws java.lang.Exception

A generic method for pruning. Must be overwritten for each new Pruningalgorithm or at least return
the Object which is passed to it.

Specified by:
postProcessRule in class PruningTemplate

Parameters:
r - - The CandidateRule to prune.
data - - The pruningset

Returns: Returns a pruned CandidateRule

Throws: java.lang.Exception

outerSplit
public weka.core.Instances[] outerSplit(weka.core.Instances data)
    throws java.lang.Exception

Does nothing
Specified by:
outerSplit in class PruningTemplate
Parameters:
data - the initial set of examples
Returns:
the initial set of examples
Throws:
java.lang.Exception

innerSplit
public weka.core.Instances[] innerSplit(weka.core.Instances data)
    throws java.lang.Exception

Does nothing
Specified by:
innerSplit in class PruningTemplate
Parameters:
data - the initial set of examples
Returns:
the initial set of examples
Throws:
java.lang.Exception

setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception

Description copied from class: PruningTemplate
Sets the options for the Pruningalgorithms.
Specified by:
setOptions in interface weka.core.OptionHandler
Overrides:
setOptions in class PruningTemplate
Parameters:
options - - The options
Throws:
java.lang.Exception

Class / Interface 45: REPruning
public class REPruning
extends PruningTemplate
Implements the REP Pruningalgorithm. Valid options are:
-D debug flag
Flag wether debug mode is enabled or not.
-P pruning flag
Flag wether pruning is enabled or not.
-N number
Set number of folds for pruning. One fold is used as the pruning set. (Default: 3)
-SH Set the heuristic for Pruning (Default: seco.heuristics.Laplace) -LC Literalstopping Criterion
Set the Literalstopping Criterion. (Default: seco.pruning.criterion.NoStopping)
-CC Clausestopping Criterion
Set the Clausestopping Criterion. (Default: seco.pruning.criterion.NoStopping)

-OP

If not specified REP uses the Operators delete-last-condition and delete-rule operator. Else it uses all Operators. find-best-simplification, delete-rule, delete-last-condition

TODO: - Specification of pruning operators. Use 1,2,...or all operators.

### Constructor Detail

#### REPruning

```java
public REPruning()
```

### Method Detail

#### postProcess

```java
public RuleSet postProcess(RuleSet rs, 
weka.core Instances data)
throws java.lang.Exception
```

This Method processes a rule set, after the theory is learned. In the case of REP it tries to find the best simplification for the RuleSet.

**Specified by:**
postProcess in class PruningTemplate

**Parameters:**
- rs - The ruleset to prune
- data - The pruning set

**Returns:**
The best simplification for a ruleset

**Throws:**
java.lang.Exception

#### postProcessRule

```java
public CandidateRule postProcessRule(CandidateRule r, 
weka.core Instances data)
throws java.lang.Exception
```

Does nothing.

**Specified by:**
postProcessRule in class PruningTemplate

**Parameters:**
- r - the rule to prune
- data - the pruning set

**Returns:**
the rule r

**Throws:**
java.lang.Exception

#### setOptions

```java
public void setOptions(java.lang.String[] options)
throws java.lang.Exception
```

Sets the options for REP

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Overrides:**
setOptions in class PruningTemplate

**Parameters:**
- options - The options String

**Throws:**
java.lang.Exception
getOptions
public java.lang.String[] getOptions()

Specified by:
getOptions in interface weka.core.OptionHandler

Overrides:
getOptions in class PruningTemplate

clauseStop
public boolean clauseStop(RuleSet rs,
CandidateRule r,
weka.core.Instances data)

Implements the ClauseStoppingCriterion. Is always false.

Specified by:
clauseStop in class PruningTemplate

Parameters:
rs - The RuleSet
r - The CandidateRule
data - The test set

Returns:
false

Throws:
java.lang.Exception

listOptions
public java.util.Enumeration listOptions()

Specified by:
listOptions in interface weka.core.OptionHandler

Overrides:
listOptions in class PruningTemplate

literalStop
public boolean literalStop(CandidateRule r,
weka.core.Instances data)

Implements the LiteralStoppingCriterion. Is always false.

Specified by:
literalStop in class PruningTemplate

Parameters:
r - The CandidateRule
data - The test set

Returns:
false

Throws:
java.lang.Exception

outerSplit
public weka.core.Instances[] outerSplit(weka.core.Instances data)

throws java.lang.Exception

Splits the examples into to set.
The growing set (Instances[0]), which contains (num_folds-1)/num_folds of the initial instance set size,
and the pruning set (Instances[1]), which contains 1/num_folds of the initial set size.
The growing set is used for learning and the pruning set is used for pruning. If num_folds is smaller
than zero then the set is not split.

Specified by:
outerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

innerSplit

public weka.core.Instances[] innerSplit(weka.core.Instances data)
throws java.lang.Exception

Does nothing

**Specified by:**
innerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
the initial set of examples

**Throws:**
java.lang.Exception

---

**Class / Interface 46: TDPruning**

public class TDPruning
extends PruningTemplate
This Class implements the TDP-Pruning Algorithm

**Constructor Detail**

TDPruning
public TDPruning()

**Method Detail**

clauseStop
public boolean clauseStop(RuleSet rs,
CandidateRule r,
weka.core.Instances data)
throws java.lang.Exception

**Description copied from class: PruningTemplate**
A generic method. Implements the ClauseStoppingCriterion. Must be overwritten for each new Pruning algorithm or at least return the Object which is passed to it.

**Specified by:**
clauseStop in class PruningTemplate

**Parameters:**
rs - - The RuleSet to prune.
data - - The test set

**Returns:**
Returns Whether to stop rule learning (true) or not (false)

**Throws:**
java.lang.Exception
instantiation

public boolean literalStop(CandidateRule r, weka.core.Instances data) throws java.lang.Exception

Description copied from class: PruningTemplate
A generic method. Implements the LiteralStoppingCriterion. Must be overwritten for each new

Specified by:
literalStop in class PruningTemplate

Parameters:
r - - The CandidateRule to prune.
data - - The test set

Returns:
Returns Whether to stop rule refinement (true) or not (false)

Throws:
java.lang.Exception

postProcess

public RuleSet postProcess(RuleSet rs, weka.core.Instances data) throws java.lang.Exception

Description copied from class: PruningTemplate
A generic method for pruning. Must be overwritten for each new Pruningalgorithm or at least return

the Object which is passed to it.

Specified by:
postProcess in class PruningTemplate

Parameters:
rs - - The RuleSet to prune.
data - - The pruningset

Returns:
Returns a pruned RuleSet

Throws:
java.lang.Exception

postProcessRule

public CandidateRule postProcessRule(CandidateRule r, weka.core.Instances data) throws java.lang.Exception

Description copied from class: PruningTemplate
A generic method for pruning. Must be overwritten for each new Pruningalgorithm or at least return

the Object which is passed to it.

Specified by:
postProcessRule in class PruningTemplate

Parameters:
r - - The CandidateRule to prune.
data - - The pruningset

Returns:
Returns a pruned CandidateRule

Throws:
java.lang.Exception

innerSplit

public weka.core.Instances[] innerSplit(weka.core.Instances data) throws java.lang.Exception

Description copied from class: PruningTemplate
the trainingexamples in the covering -loop Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size, and the pruning set (Instances[1]), which contains 1/3 of the initial set size. The growing set is used for learning and the pruning set is used for pruning

**Specified by:**
innerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

```java
public weka.core.Instances[] outerSplit(weka.core.Instances data)
    throws java.lang.Exception
```

**Description copied from class: PruningTemplate**
Splits the examples into two sets. The growing set (Instances[0]), which contains 2/3 of the initial instance set size, and the pruning set (Instances[1]), which contains 1/3 of the initial set size. The growing set is used for learning and the pruning set is used for pruning

**Specified by:**
outerSplit in class PruningTemplate

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

```java
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
```

**Description copied from class: PruningTemplate**
Sets the options for the Pruning algorithms.

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Overrides:**
setOptions in class PruningTemplate

**Parameters:**
options - The options

**Throws:**
java.lang.Exception
C6 – Paket „seco.pruning.criterion“

<table>
<thead>
<tr>
<th>Klasse</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>CutOff</td>
<td>CutOff Stoppingcriterion.</td>
</tr>
<tr>
<td>MaximumErrorRate</td>
<td>The MaximumErrorRate StoppingCriterion.</td>
</tr>
<tr>
<td>MDL</td>
<td>The MDL-Stoppingcriterion based on Bernhard Pfahringer MDL-formula.</td>
</tr>
<tr>
<td>NoStopping</td>
<td>The StoppingCriterion which doesn't stop anything.</td>
</tr>
<tr>
<td>RelativeCutOff</td>
<td>CutOff Stoppingcriterion.</td>
</tr>
<tr>
<td>Significance</td>
<td>The Significance Test.</td>
</tr>
</tbody>
</table>

Anhang C - 7: Paketinhalt des Pakets ‚seco.pruning.criterion’

Class / Interface 47: CutOff

```
public class CutOff
    extends Criterion
    implements IRuleStoppingCriterion, IStoppingCriterion

CutOff Stoppingcriterion.
Performs the CutOff based on an absolute evaluation of a rule.

Constructor Detail

CutOff
public CutOff()

Method Detail

clauseStop
public boolean clauseStop(RuleSet rs,
                        CandidateRule r,
                        weka.core.Instances data)
    throws java.lang.Exception
Decides when to stop adding rules to a ruleset, based on an absolute CutOff.

Specified by:
clauseStop in interface IRuleStoppingCriterion

Parameters:
rs - The ruleset
r - The rule for which the CutOff is decided
data - Can be ignored here.

Returns:
ture, iff the rules value is larger than the given threshold.

Throws:
java.lang.Exception

literalStop
public boolean literalStop(CandidateRule r,
                          weka.core.Instances data)
    throws java.lang.Exception
Decides when to stop refining a rule.

Specified by:
literalStop in interface IStoppingCriterion

Parameters:
r - The rule for which the CutOff is decided
data - Can be ignored here.

Returns:
ture, iff the rules value is larger than the given threshold.
```
Class / Interface 48: MaximumErrorRate

public class MaximumErrorRate extends Criterion
implements IRuleStoppingCriterion, IStoppingCriterion
The MaximumErrorRate StoppingCriterion. Decides when to stop adding rules to a ruleset, based on a rules error rate. Also decides when to stop refining a rule based on the rules error rate.

Constructor Detail

MaximumErrorRate
public MaximumErrorRate()

Method Detail

clauseStop
public boolean clauseStop(RuleSet rs,
                     CandidateRule r,
                     weka.core.Instances data)
    throws java.lang.Exception
Decides when to stop adding rules to a ruleset, based on a rules error rate.

Specified by:
clauseStop in interface IRuleStoppingCriterion

Parameters:
rs - The ruleset
r - The rule for which the error is estimated
data - the data on which the error is calculated.

Returns:
true, iff the rules error is larger than 50%

Throws:
java.lang.Exception

literalStop
public boolean literalStop(CandidateRule r,
                           weka.core.Instances data)
    throws java.lang.Exception

Specified by:
literalStop in interface IStoppingCriterion

Parameters:
r - The rule for which the error is estimated
data - the data on which the error is calculated.

Returns:
true, iff the rules error is larger than 50%

Throws:
java.lang.Exception
public class **MDL**
extends Criterion
implements IRuleStoppingCriterion, IStoppingCriterion


### Constructor Detail

**MDL**

```java
public MDL()
```

### Method Detail

```java
public boolean clauseStop(RuleSet rs, CandidateRule r, weka.core.Instances data)
```

Decides when to stop adding rules to a ruleset, based on a MDL-Criterion.

**Specified by:**

clauseStop in interface IRuleStoppingCriterion

**Parameters:**
- rs - The ruleset
- r - The rule
- data - the data which is tried to compress

**Returns:**
- true, iff \( L(\text{rule}) + L(\text{data}|\text{rule}) < L(\text{data}) \)

**Throws:**
- java.lang.Exception

```java
public boolean literalStop(CandidateRule r, weka.core.Instances data)
```

Decides when to stop refining rules based on a MDL-Criterion.

**Specified by:**

literalStop in interface IStoppingCriterion

**Parameters:**
- r - The rule
- data - the data which is tried to compress

**Returns:**
- true, iff \( L(\text{rule}) + L(\text{data}|\text{rule}) < L(\text{data}) \)

**Throws:**
- java.lang.Exception

---

**Class / Interface 50: NoStopping**

```java
public class NoStopping
extends Criterion
implements IRuleStoppingCriterion, IStoppingCriterion
```

The StoppingCriterion which doesn't stop anything. All Methods return false.
### Constructor Detail

#### NoStopping

```java
public NoStopping()
```

### Method Detail

#### clauseStop

```java
public boolean clauseStop(RuleSet rs, CandidateRule r, weka.core.Instances data)
throws java.lang.Exception
```

**Description copied from interface: IRuleStoppingCriterion**

Decides when to stop adding rules to a ruleset.

**Specified by:**

clauseStop in interface IRuleStoppingCriterion

**Parameters:**
- rs - The Ruleset
- r - The rule
- data - The data

**Returns:**
true iff the criterion is satisfied

**Throws:**
java.lang.Exception

#### literalStop

```java
public boolean literalStop(CandidateRule r, weka.core.Instances data)
throws java.lang.Exception
```

**Specified by:**

literalStop in interface IStoppingCriterion

**Parameters:**
- r - The rule
- data - The data

**Returns:**
true iff the criterion is satisfied

**Throws:**
java.lang.Exception

### Class / Interface 51: RelativeCutOff

```java
public class RelativeCutOff extends Criterion implements IRuleStoppingCriterion, IStoppingCriterion
```

Performs the CutOff based on a relative evaluation of a rule. The rule is compared to its direct predecessor rule.

### Constructor Detail

#### RelativeCutOff

```java
public RelativeCutOff()
```
Method Detail

clauseStop
public boolean clauseStop(RuleSet rs,
    CandidateRule r,
    weka.core.Instances data)
    throws java.lang.Exception
Decides when to stop adding rules to a ruleset, based on an absolute CutOff.
Specified by:
clauseStop in interface IRuleStoppingCriterion
Parameters:
rs - The ruleset
r - The rule for which the CutOff is decided
data - Can be ignored here.
Returns:
true, iff the rules value is larger than the given threshhold.
Throws:
throws - an Exception, because it is not yet implemented
java.lang.Exception

literalStop
public boolean literalStop(CandidateRule r,
    weka.core.Instances data)
    throws java.lang.Exception
Specified by:
literalStop in interface IStoppingCriterion
Parameters:
r - The rule for which the CutOff is decided
data - Can be ignored here.
Returns:
true, iff the rules value is larger than the given threshhold.
Throws:
java.lang.Exception

Class / Interface 52: Significance
public class Significance
extends Criterion
implements IRuleStoppingCriterion, IStoppingCriterion
The Significance Test.

Constructor Detail

Significance
public Significance()

Method Detail

clauseStop
public boolean clauseStop(RuleSet rs,
    CandidateRule r,
    weka.core.Instances data)
    throws java.lang.Exception
decides whether the likelihood ratio of a ruleset exceeds the threshold or not

**Specified by:**
clauseStop in interface IRuleStoppingCriterion

**Parameters:**
- r - The rule
- data - The data
- rs - The Ruleset

**Returns:**
true, iff the likelihood ratio exceeds the threshold

**Throws:**
java.lang.Exception

---

literalStop

public boolean literalStop(CandidateRule r, weka.core.Instances data)
decides whether the likelihood ratio of a rule exceeds the threshold or not

**Specified by:**
literalStop in interface IStoppingCriterion

**Parameters:**
- r - The rule
- data - The data

**Returns:**
true, iff the likelihood ratio exceeds the threshold

---

getOptions

public java.lang.String[] getOptions()

**Specified by:**
getOptions in interface weka.core.OptionHandler

**Overrides:**
getOptions in class Criterion

---

setOptions

public void setOptions(java.lang.String[] options)
throws java.lang.Exception

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Overrides:**
setOptions in class Criterion

**Throws:**
java.lang.Exception
### C7 – Paket „seco.pruning.model“

<table>
<thead>
<tr>
<th>Klassen/Interfaces</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRuleStoppingCriterion</td>
<td>Offers the Methods needed for the RuleStoppingCriterion</td>
</tr>
<tr>
<td>IStoppingCriterion</td>
<td>Offers the Methods needed for the StoppingCriterion</td>
</tr>
<tr>
<td>PruningTemplate</td>
<td>Template Class for Pruning behavior in the Covering class.</td>
</tr>
<tr>
<td>RuleOperator</td>
<td>This is a generic class for Ruleoperators</td>
</tr>
<tr>
<td>RuleSetOperator</td>
<td>This is a generic class for Rulesetoperators</td>
</tr>
<tr>
<td>Criterion</td>
<td>This is a generic class for a stopping criterion.</td>
</tr>
</tbody>
</table>

Anhang C - 8: Paketinhalte des Pakets "seco.pruning.model"

#### Class / Interface 53: IRuleStoppingCriterion

**public interface IRuleStoppingCriterion**

Offers the Methods needed for the RuleStoppingCriterion

Method Detail

clauseStop

```java
boolean clauseStop(RuleSet rs,
    CandidateRule r,
    weka.core.Instances data)
```

throws java.lang.Exception

Decides when to stop adding rules to a ruleset.

**Parameters:**
- rs - The Ruleset
- r - The rule
- data - The data

**Returns:**
- true iff the criterion is satisfied

**Throws:**
- java.lang.Exception

#### Class / Interface 54: IStoppingCriterion

**public interface IStoppingCriterion**

Offers the Methods needed for the StoppingCriterion

Method Detail

literalStop

```java
boolean literalStop(CandidateRule r,
    weka.core.Instances data)
```

throws java.lang.Exception

**Parameters:**
- r - The rule
- data - The data

**Returns:**
- true iff the criterion is satisfied

**Throws:**
- java.lang.Exception
### Class / Interface 55: Criterion

```java
public abstract class Criterion
extends java.lang.Object
implements java.io.Serializable, weka.core.OptionHandler
```

This is a generic class for a stopping criterion. It is in fact only a container class, which holds the options for a Criterion.

Possible options are:

- **-TH** - Sets the threshold for a Stopping criterion (Default: 0)

- **-SH** - Sets the Search heuristic for a Stopping criterion (Default: seco.heuristics.Laplace)

### Constructor Detail

**Criterion**

```java
public Criterion()
```

### Method Detail

**setHeuristic**

```java
public void setHeuristic(SearchHeuristic h)
```

**setThreshold**

```java
public void setThreshold(double th)
```

**getHeuristic**

```java
public SearchHeuristic getHeuristic()
```

**getThreshold**

```java
public double getThreshold()
```

**getOptions**

```java
public java.lang.String[] getOptions()
```

**toOptionString**

```java
public java.lang.String toOptionString()
```

#### Specified by:
getOptions in interface weka.core.OptionHandler

toOptionString in interface weka.core.OptionHandler

**listOptions**

```java
public java.utilEnumeration listOptions()
```

**setOptions**

```java
public void setOptions(java.lang.String[] options)
```

#### Specified by:
setOptions in interface weka.core.OptionHandler
### Throws:
java.lang.Exception

### Class / Interface 56: PruningTemplate
public abstract class **PruningTemplate**
extends java.lang.Object
implements weka.core.OptionHandler, java.lang.Cloneable, java.io.Serializable
Template Class for Pruning behavior in the Covering class. Valid options are:

- **-D** debug flag
- **-P** pruning flag
- **-N** Set number of folds for pruning. One fold is used as the pruning set. (Default: 3)
- **-SH** Set the heuristic for Pruning (Default: seco.heuristics.Laplace) -LC Literalstopping Criterion
- **-CC** Clausestopping Criterion
- **-LC** Literalstopping Criterion
- **-RE** Set the number of restarts. (Default: 0)

### Constructor Detail
PruningTemplate
public **PruningTemplate**()  
A Standard constructor

### Method Detail
#### postProcess
public abstract RuleSet **postProcess**(RuleSet rs,  
weka.core.Instances data)  
throws java.lang.Exception
A generic method for pruning. Must be overwritten for each new Pruning algorithm or at least return the Object which is passed to it.

**Parameters:**
rs - - The RuleSet to prune.
data - - The pruning set

**Returns:**
Returns a pruned RuleSet

**Throws:**
java.lang.Exception

#### postProcessRule
public abstract CandidateRule **postProcessRule**(CandidateRule r,  
weka.core.Instances data)  
throws java.lang.Exception
A generic method for pruning. Must be overwritten for each new Pruning algorithm or at least return the Object which is passed to it.

**Parameters:**
r - - The CandidateRule to prune.
data - - The pruningset

Returns: 
Returns a pruned CandidateRule

Throws: 
java.lang.Exception

---
literalStop
public abstract boolean literalStop(CandidateRule r, 
    weka.core.Instances data) 
throws java.lang.Exception

A generic method. Implements the LiteralStoppingCriterion. Must be overwritten for each new

Parameters: 
r - - The CandidateRule to prune.
data - - The test set

Returns: 
Returns Whether to stop rule refinement (true) or not (false)

Throws: 
java.lang.Exception

---
clauseStop
public abstract boolean clauseStop(RuleSet rs, 
    CandidateRule r, 
    weka.core.Instances data) 
throws java.lang.Exception

A generic method. Implements the ClauseStoppingCriterion. Must be overwritten for each new

Pruningalgorithm or at least return the Object which is passed to it.

Parameters: 
rs - - The RuleSet to prune.
data - - The test set

Returns: 
Returns Whether to stop rule learning (true) or not (false)

Throws: 
java.lang.Exception

---
innerSplit
public abstract weka.core.Instances[] innerSplit(weka.core.Instances data) 
throws java.lang.Exception

the trainingexamples in the covering -loop Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size, 
and the pruning set (Instances[1]), which contains 1/3 of the initial set size.
The growing set is used for learning and the pruning set is used for pruning

Parameters: 
data - the initial set of examples

Returns: 
instances[0]: the growing set; 
instances[1]: the pruning set

Throws: 
java.lang.Exception

---
outerSplit
public abstract weka.core.Instances[] outerSplit(weka.core.Instances data) 
throws java.lang.Exception

Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size,
and the pruning set (Instances[1]), which contains 1/3 of the initial set size.
The growing set is used for learning and the pruning set is used for pruning

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

split
public static weka.core.Instances[] split(weka.core.Instances data, int num_folds) throws java.lang.Exception
Splits the examples into to set.
The growing set (Instances[0]), which contains (num_folds-1)/num_folds of the initial instance set size,
and the pruning set (Instances[1]), which contains 1/num_folds of the initial set size.
The growing set is used for learning and the pruning set is used for pruning. If num_folds is smaller
than zero then the set is not split.

**Parameters:**
data - the initial set of examples
num_folds - the number of folds used for splitting

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

split
public static weka.core.Instances[] split(weka.core.Instances data) throws java.lang.Exception
Splits the examples into to set.
The growing set (Instances[0]), which contains 2/3 of the initial instance set size,
and the pruning set (Instances[1]), which contains 1/3 of the initial set size.
The growing set is used for learning and the pruning set is used for pruning

**Parameters:**
data - the initial set of examples

**Returns:**
instances[0]: the growing set;
instances[1]: the pruning set

**Throws:**
java.lang.Exception

---

calculateRuleStats
public CandidateRule calculateRuleStats(CandidateRule r, weka.core.Instances data) throws java.lang.Exception
Copies a CandidateRule and calculates the TwoClassStats on the given data.

**Parameters:**
r - The CandidateRule
data - The given data

**Throws:**
java.lang.Exception
getOptions
public java.lang.String[] getOptions()

Specified by:
getOptions in interface weka.core.OptionHandler

listOptions
public java.util.Enumeration listOptions()

Specified by:
listOptions in interface weka.core.OptionHandler

setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception
Sets the options for the Pruning algorithms.

Specified by:
setOptions in interface weka.core.OptionHandler

Parameters:
options - The options

Throws:
java.lang.Exception

getHeuristic
public SearchHeuristic getHeuristic()

Returns:
the m_heuristic

setHeuristic
public void setHeuristic(SearchHeuristic m_heuristic)

Parameters:
m_heuristic - the m_heuristic to set

getNumFolds
public int getNumFolds()

Returns:
the m_setFolds

setNumFolds
public void setNumFolds(int folds)

Parameters:
folds - the m_setFolds to set

toOptionString
public java.lang.String toOptionString()

Returns:
a string representation of the Pruning algorithm, including all parameters (if any). If there are
parameters, the returned string will start and end with quotes. Thus the representation is suitable for
the command-line (e.g., for initializing other objects).

isRestartEnabled
public boolean isRestartEnabled()

Returns:
the m_restartEnabled

setRestartEnabled
public void setRestartEnabled(boolean enabled)
Parameters:
enabled - the m_restartEnabled to set

getNumRestarts
public int getNumRestarts()
Returns:
the m_numRestarts

setNumRestarts
public void setNumRestarts(int restarts)
Parameters:
restarts - the m_numRestarts to set

Class / Interface 57: RuleOperator
public abstract class RuleOperator
extends java.lang.Object
implements java.io.Serializable, weka.core.OptionHandler
This is a generic class for Ruleoperators Valid options are:
-D debug flag
Flag wether debug mode is enabled or not.
-SH Set the heuristic for Pruning (Default: seco.heuristics.Laplace)

Constructor Detail
RuleOperator
public RuleOperator() A defaul constructor

Method Detail
applyOperator
public abstract CandidateRule applyOperator(CandidateRule r,
weka.core.Instances data)
throws java.lang.Exception
This Method applies a specific Pruningoperator to a CandidateRule
Parameters:
r - The rule to prune
data - The pruning set
Returns:
The pruned rule or null if the operator cannot be applied
Throws:
java.lang.Exception

getHeuristic
public SearchHeuristic getHeuristic()
Returns:
the m_heuristic

setHeuristic
public void setHeuristic(SearchHeuristic m_heuristic)
Parameters:
m_heuristic - the m_heuristic to set
getOptions
public java.lang.String[] getOptions()

Specified by:
getOptions in interface weka.core.OptionHandler

listOptions
public java.util.Enumeration listOptions()

Specified by:
listOptions in interface weka.core.OptionHandler

setOptions
public void setOptions(java.lang.String[] options)
    throws java.lang.Exception

Specified by:
setOptions in interface weka.core.OptionHandler

Throws:
java.lang.Exception

Class / Interface 58: RuleSetOperator

public abstract class RuleSetOperator
extends java.lang.Object
implements java.io.Serializable, weka.core.OptionHandler
This is a generic class for RuleSet operators. A RuleSetOperator applied to a RuleSet returns a set of new RuleSets. The operator is applied to each rule in the ruleset and creates a new RuleSet with one transformed rule.

1: Theories = empty
2: For Each Rule r in RuleSet rs
3:   r' = apply Operator to r
4:   rs' = rs \ r
5:   rs' = rs' + r'
6:   Theories = Theories + rs'
7: return(Theories)

Valid options are:
-D debug flag
Flag wether debug mode is enabled or not.
-SH Set the heuristic for Pruning (Default: seco.heuristics.Laplace)

Constructor Detail

RuleSetOperator
public RuleSetOperator()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getOptions()</code></td>
<td>Returns a set of RuleSets</td>
</tr>
<tr>
<td><code>setOptions(String[] options)</code></td>
<td>Sets the heuristic</td>
</tr>
<tr>
<td><code>getHeuristic()</code></td>
<td>Gets the SearchHeuristic</td>
</tr>
<tr>
<td><code>setHeuristic(SearchHeuristic heur)</code></td>
<td>Sets the SearchHeuristic</td>
</tr>
</tbody>
</table>

**Parameters:**
- `rs` - The Ruleset which the operator is applied to
- `data` - The pruning data in case needed

**Returns:**
A set of RuleSets

**Throws:**
java.lang.Exception

**Specified by:**
getOptions in interface weka.core.OptionHandler

**Specified by:**
listOptions in interface weka.core.OptionHandler

**Specified by:**
setOptions in interface weka.core.OptionHandler

**Throws:**
java.lang.Exception

**Specified by:**
getHeuristic in interface weka.core.SearchHeuristic

**Specified by:**
setHeuristic in interface weka.core.SearchHeuristic

**Parameters:**
- `heur` - the m_heuristic to set
C8 – Paket „seco.pruning.operator“

<table>
<thead>
<tr>
<th>Klassennamen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleDeleteLastCondition</td>
<td>This is a Rule operator, which deletes the last condition of a rule.</td>
</tr>
<tr>
<td>RuleFindBestSimplification</td>
<td>This is a rule operator, which searches for the best simplification of a rule given the data.</td>
</tr>
<tr>
<td>RuleFindBestReplacement</td>
<td>This rule operator tries to find the best replacement of a literal in a rule.</td>
</tr>
<tr>
<td>RuleIdentity</td>
<td>This is a rule operator which does nothing.</td>
</tr>
<tr>
<td>RuleSetDeleteLastCondition</td>
<td>This Ruleset operator returns a set of RuleSets.</td>
</tr>
<tr>
<td>RuleSetDeleteRule</td>
<td>This Ruleset operator returns a set of RuleSets.</td>
</tr>
<tr>
<td>RuleSetFindBestSimplification</td>
<td>This Ruleset operator returns a set of RuleSets.</td>
</tr>
<tr>
<td>RuleSetIdentity</td>
<td>This Ruleset operator returns a set containing one RuleSet.</td>
</tr>
</tbody>
</table>

Anhang C - 9: Paketinhalt des Pakets ”seco.pruning.operator“

**Class / Interface 59: RuleDeleteLastCondition**

```java
public class RuleDeleteLastCondition
    extends RuleOperator
    implements java.io.Serializable

This is a Rule operator, which deletes the last condition of a rule.
```

**Constructor Detail**

```java
public RuleDeleteLastCondition()
```

**Method Detail**

```java
applyOperator
public CandidateRule applyOperator(CandidateRule r, weka.core.Instances data)
    throws java.lang.Exception
```

**Description copied from class: RuleOperator**

This Method applies a specific Pruningoperator to a CandidateRule

**Specified by:**

applyOperator in class RuleOperator

**Parameters:**

- r - The rule to prune
- data - The pruning set

**Returns:**

- a candidateRule with the last condition deleted or r if the operator cant be applied

**Throws:**

- java.lang.Exception

**Class / Interface 60: RuleFindBestReplacement**

```java
public class RuleFindBestReplacement
    extends RuleOperator

This rule operator tries to find the best replacement of a literal in a rule. not yet implemented properly.
```

**Todo:** Implementation work.
Constructor Detail

RuleFindBestReplacement
public RuleFindBestReplacement()

Method Detail

applyOperator
public CandidateRule applyOperator(CandidateRule r,
weka.core.Instances data)
throws java.lang.Exception

Description copied from class: RuleOperator
This Method applies a specific Pruningoperator to a CandidateRule

Specified by:
applyOperator in class RuleOperator

Parameters:
r - The rule to prune
data - The pruning set

Returns:
The pruned rule or null if the operator cannot be applied

Throws:
java.lang.Exception

Class / Interface 61: RuleFindBestSimplification

public class RuleFindBestSimplification
extends RuleOperator
This is a rule operator, which searches for the best simplification of a rule given the data.

Constructor Detail

RuleFindBestSimplification
public RuleFindBestSimplification()

Method Detail

applyOperator
public CandidateRule applyOperator(CandidateRule r,
weka.core.Instances data)
throws java.lang.Exception

Description copied from class: RuleOperator
This Method applies a specific Pruningoperator to a CandidateRule

Specified by:
applyOperator in class RuleOperator

Parameters:
r - The rule to be pruned
data - The data for the search of the best simplification

Returns:
the best simplification of the rule r or the r itself if no simplification is found

Throws:
java.lang.Exception
**Class / Interface 62: RuleIdentity**

public class **RuleIdentity**
extends RuleOperator
implements java.io.Serializable
This is a rule operator which does nothing. It returns a rule without changing it

**Constructor Detail**

RuleIdentity
public **RuleIdentity**()

**Method Detail**

applyOperator
public CandidateRule **applyOperator**(CandidateRule r,
weka.core.Instances data)
throws java.lang.Exception

*Description copied from class: RuleOperator*
This Method applies a specific Pruningoperator to a CandidateRule

*Specified by:*
applyOperator in class RuleOperator

*Parameters:*
r - The Rule to be pruned
data - The data on which the rule is pruned

*Returns:*
The rule

*Throws:*
java.lang.Exception

**Class / Interface 63: RuleSetDeleteLastCondition**

public class **RuleSetDeleteLastCondition**
extends RuleSetOperator
This RuleSet operator generates a set of new RuleSets. For each rule the last condition is deleted an a
new Ruleset is generated

**Constructor Detail**

RuleSetDeleteLastCondition
public **RuleSetDeleteLastCondition**()

**Method Detail**

applyOperator
public java.util.Vector<RuleSet> **applyOperator**(RuleSet rs,
weka.core.Instances data)
throws java.lang.Exception

*Description copied from class: RuleSetOperator*
Returns a set of rulesets where every Rule is simplified according to the operator

*Specified by:*
applyOperator in class RuleSetOperator
Parameters:
rs - The RuleSet to be pruned
data - The data on which the ruleset is pruned

Returns:
A set of new RuleSets

Throws:
java.lang.Exception

Class / Interface 64: RuleSetDeleteRule
public class RuleSetDeleteRule
extends RuleSetOperator
This Ruleset operator returns a set of RuleSets. Each rule is deleted and a new Ruleset is generated

Constructor Detail
RuleSetDeleteRule
public RuleSetDeleteRule()

Method Detail
applyOperator
public java.util.Vector<RuleSet> applyOperator(RuleSet rs,
                                               weka.core.Instances data)
                                               throws java.lang.Exception
Description copied from class: RuleSetOperator
Returns a set of rule sets where every Rule is simplified according to the operator

Specified by:
applyOperator in class RuleSetOperator

Parameters:
rs - The RuleSet to be pruned
data - The data on which the ruleset is pruned

Returns:
A set of new RuleSets

Throws:
java.lang.Exception

Class / Interface 65: RuleSetFindBestSimplification
public class RuleSetFindBestSimplification
extends RuleSetOperator
This Ruleset operator returns a set of RuleSets. For each rule it is tried to find the best simplification and a new Ruleset is created

Constructor Detail
RuleSetFindBestSimplification
public RuleSetFindBestSimplification()
Method Detail

applyOperator
public java.util.Vector<RuleSet> applyOperator(RuleSet rs, weka.core.Instances data)
throws java.lang.Exception

Description copied from class: RuleSetOperator
Returns a set of rulesets where every Rule is simplified according to the operator

Specified by:
applyOperator in class RuleSetOperator

Parameters:
rs - The RuleSet to be pruned
data - The data on which the ruleset is pruned

Returns:
A set of new RuleSets

Throws:
java.lang.Exception

setOptions
public void setOptions(java.lang.String[] options)
throws java.lang.Exception

Specified by:
setOptions in interface weka.core.OptionHandler

Overrides:
setOptions in class RuleSetOperator

Throws:
java.lang.Exception

Class / Interface 66: RuleSetIdentity

public class RuleSetIdentity
extends RuleSetOperator
This RuleSet operator returns a Set with one RuleSet.

Constructor Detail

RuleSetIdentity
public RuleSetIdentity()
rs - The Ruleset to be pruned
data - The data on which the ruleset is pruned

**Returns:**
A Set containing the RuleSet rs

**Throws:**
java.lang.Exception