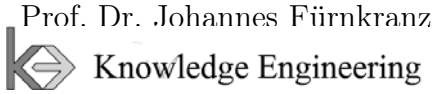
Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Knowledge Engineering
Prof. Dr. Johannes Fürnkranz



Vergleich von AQ, CN2 und CN2 mit Weighted Covering

Diplomarbeit

Marc Ruppert

Betreuung durch
Prof. Dr. Johannes Fürnkranz

Dezember 2006

Zusammenfassung

Diese Arbeit beginnt mit einer Einführung in das maschinelle Lernen. Das Lernproblem wird definiert und die Problemlösung mittels **Separate-and-Conquer** (SeCo) Algorithmen vorgestellt. Der AQ und der CN2-Algorithmus sind zwei Separate and Conquer Algorithmen, die in dieser Arbeit genauer vorgestellt werden. Das **SeCo-Framework** findet Verwendung in dieser Arbeit und die Implementierung neuer Algorithmen wird anhand des AQ-Algorithmus erklärt. Die Idee des **Weighted Covering** wird hier aufgefasst und die Umsetzung in der SeCo-Factory aufgezeigt. Anschließend werden durch die Evaluierung die Algorithmen der SeCo-Factory mit Standardeinstellungen verglichen und die eventuellen Vorteile und Nachteile des Weighted Covering näher beleuchtet. Abschließend erfolgt ein Vergleich der Algorithmen mit verschiedenen Parametern und Weighted Covering.

Ehrenwörtliche 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 den 21.12.2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Definitionen

Quelltexte

1 Einleitung

1.1 Einführung

Was ist *Maschinelles Lernen*? Von Maschinellem Lernen spricht man, wenn mit Maschinen aus Erfahrung Wissen generiert wird. Es wird versucht in bekannten Daten Gesetzmäßigkeiten zu finden, um z.B. neue Daten beurteilen zu können. So könnte ein Problem aussehen: „Lerne Backgammon zu spielen“. Um das zu erreichen werden meist Algorithmen genutzt, die sich in verschiedene Klassen einteilen lassen. Davon betrachten wir nur die Algorithmen, die zur Klasse der Separate-and-Conquer Algorithmen gezählt werden. Es gibt eine Vielzahl von Separate-and-Conquer Algorithmen, die unterschiedliche *Eigenschaften* und *Vorgehensweisen* haben, deren *Grundstruktur* aber gleich ist. Die Algorithmen werden durch Vergleiche beurteilt, wobei die gelernten Regelmengen gegenübergestellt und hinsichtlich der Korrektheit und Größe verglichen werden. In der Regel geht ein Algorithmus als Sieger hervor und wird dem anderen vorgezogen. Es gibt aber nicht immer eindeutige Sieger über alle Trainingsdaten, es müssen dann genau die Fälle unterschieden werden, in welchen ein Algorithmus besser als ein anderer ist. Es muss unterschieden werden, auf welchen Daten ein Algorithmus besser als ein anderer ist. Es gibt meist wichtige Eigenschaften, die den speziellen Charakter des jeweiligen Algorithmus ausmachen. Das genaue Messen dieser Ausmaße wird dadurch erschwert, dass z.B. die Implementation der Algorithmen völlig unterschiedlich ist, da es sich um verschiedene Autoren handelt. Deswegen wird im Rahmen dieser Arbeit auf das Seco-Frame-Work [1] zurückgegriffen. Dieses erlaubt die SeCo-Algorithmen (Separate and Conquer Algorithmen) modular zu implementieren, was die Untersuchung einzelner Merkmale vereinfacht. Da die Beschreibung der Algorithmen in XML erfolgt, ist eine Modifikation der Algorithmen ohne Veränderung des Quelltextes und somit ohne erneutes Übersetzen möglich. Im Rahmen dieser Arbeit wird der AQ Algorithmus aus [3] mit der CN2 Variante von Bexa, welche durch das SeCo-Framework bereitgestellt wird, verglichen. Genauer handelt es sich um den AQR-Algorithmus, eine Variante des originalen AQ. Außerdem wird das Seco-Frame-Work im Rahmen dieser Arbeit um die Möglichkeit des Weighted Coverings erweitert. Damit können nun auch AQ, Bexa und CN2 mit Weighted Covering ausgestattet werden.

1.2 Überblick

Diese Arbeit beginnt mit einer Einführung in das **maschinelle Regellernen** in Kapitel 2 und die Lösung dieses Problems mit Hilfe von **Separate-and-Conquer Algorithmen**. Dabei wird der Separate-and-Conquer Algorithmus mit Hilfe von Pseudo-Code beschrieben.

Die verwendeten Algorithmen **AQ** und **CN2** werden in Kapitel 3 vorgestellt. In Kapitel 3.1.1 wird dann auf die Besonderheiten des AQ Algorithmus eingegangen.
Auf die SeCo-Factory und ihre Besonderheiten wird in Kapitel 4 eingegangen. Auch die Konfiguration eines Algorithmus mit Hilfe der SeCo-Factory wird an dem Beispiel des AQ-Algorithmus in Kapitel 4.3 verdeutlicht.
In Kapitel 5 wird auf die Möglichkeit des **Weighted Covering** eingegangen und die SeCo-Factory um die Möglichkeit erweitert, Weighted Covering für Algorithmen zu nutzen.
Die Auswertung findet sich schließlich in Kapitel 6. Im Anhang befinden sich dann noch ausführliche Auswertungen der Ergebnisse der einzelnen Algorithmen.

2 Lern-Algorithmen

Um Wesen und Tätigkeit der Lern-Algorithmen zu erklären, gehe ich an dieser Stelle zunächst auf die Frage ein, was denn das **Lernproblem** ist. Zuerst werden einige Begriffe definiert und anschließend die Familie der **Separate und Conquer** Algorithmen dargestellt[1].

2.1 Lernen

Was ist Lernen? In der Literatur hat sich bis jetzt noch keine allgemein gültige Definition des Begriffes Lernen herausgebildet. Die meisten Menschen haben eine Vorstellung von dem Lernen, haben aber Probleme klar zu definieren was Lernen ist. Deswegen hier zwei etwas neuere Definitionen des Begriffes:

> „Learning denotes changes in the system that ... enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.“ [Simon, 1983]

Andere Definition von Lernen:

Gegeben:

- ein Ziel-Konzept
- positive und negative Beispiele
- beschrieben durch verschiedene Attribute
- optionales Hintergrundwissen

Gesucht:

- eine einfaches Regel-Set, das zwischen noch ungelernten positiven und negativen Beispielen des Ziel-Konzepts unterscheiden kann

Abbildung 2.1: Das induktive Konzept-Lern-Problem (Übersetzt aus [2])

Was soll gelernt werden? Es sollen Beschreibungen für *Lernbegriffe*, auch Klassen genannt, gelernt werden. Wie z.B.: „Ich gehe morgen Golf spielen“. Um solche Lernbegriffe erlernen zu können, braucht man eine Menge an Beispielen. Diese Beispiele werden *Instanzen* genannt. Eine Menge von Beispielen ergibt die

[1] Das ganze Kapitel orientiert sich an Kapitel 2 aus [1]

Trainingsmenge, aus der Regeln gelernt werden sollen. Ein Beispiel besteht normalerweise aus mehreren *Attributen* und genau einer **Klasse**, zu der das Beispiel gehört. Ein Überblick über das generelle Entstehen von Theorien und die Klassifikation neuer Daten geben die Abbildungen 2.2 und 2.3.

2.1.1 Begriffs-Definitionen

Beginnen wir mit folgenden Definitionen (frei nach [1] und [2]):

Definition 2.1 (Nominal) *Als Nominal bezeichnen wir alle Ausprägungen von Merkmalen, die sich zwar unterscheiden, aber die Eigenschaft haben, sich nicht in eine Rangfolge bringen zu lassen.*

Beispiele für nominale Werte: **sonnig, windig, Ja, Nein, Gelb, Buch**

Definition 2.2 (Numerisch) *Als numerisch bezeichnen wir alle Ausprägungen von Merkmalen, die durch Zahlen ausgedrückt werden. Numerische Werte sind meist reelle Zahlen. Somit kann jede messbare Größe dargestellt werden. Numerische Werte besitzen eine Ordnungsfunktion, d.h. sie können miteinander verglichen und geordnet werden.*

Beispiele für numerische Werte: **42, 34.5, 0.03, -12.0, -100**

Definition 2.3 (Attribut) *Das Attribut A besteht aus drei Teilen*

- ***Name**: sollte eindeutig gewählt sein*
- ***Typ**: es gibt **nominale** und **numerische** Attribute*
- ***Werte-Menge**[2]: ist eine Menge ($\mathcal{D}_A$), die angibt, welche Werte vorkommen können.*

Als Beispiel folgende Attribute:

Tabelle 2.1: Beispiele für Attribute

Name	**Typ**	**Wertemenge**
outlook	nominal	sunny, overcast, rainy
temperature	numerisch	-15..50
class	nominal	unacc, acc, good, vgood

Die Menge von Attributen, die in einem Lernproblem existieren, bezeichnen wir als $\mathcal{A}$.

[2]Eine Werte-Menge wird auch als *Domäne* bezeichnet.

Definition 2.4 (Instanz) *Eine Instanz (auch Beispiel genannt) I ist eine Menge von Wertzuweisungen für Attribute $A \in \mathcal{A}$ der Form $A = x$, wobei $x \in \mathcal{D}_A$. Dabei wird jedem Attribut A höchstens ein Wert zugewiesen. Bringt man die Attribute in eine feste Reihenfolge $A_1, ..., A_n$, so kann man eine Instanz I auch als Kreuzprodukt $I \in \mathcal{D}_{A_1} \times ... \times \mathcal{D}_{A_n}$ darstellen, was einer Liste von Attributwerten entspricht.*

Definition 2.5 (Trainingsmenge) *Die Trainingsmenge TS ist eine gegebene Menge von Instanzen und deren Zuordnungen zu einem Begriff: $TS \in \mathcal{D}_{A_1} \times ... \times \mathcal{D}_{\mathcal{A}_\setminus} \times C$, wobei C die Menge der zu lernenden Klassen ist.*

Hier ein Beispiel für eine Trainingsmenge, bei der gelernt werden soll, ob man Golf spielen geht (yes) oder nicht (no):

Tabelle 2.2: Beispiel für eine Trainingsmenge

#	outlook	temperature	humidity	windy	play
1	overcast	hot	high	FALSE	yes
2	rainy	mild	high	FALSE	yes
3	rainy	cool	normal	FALSE	yes
4	sunny	hot	high	FALSE	no
5	sunny	hot	high	TRUE	no
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes

Definition 2.6 (Klasse) *Eine Klasse ist eine Gruppe von Beispielen.*

Meist haben diese Beispiele gemeinsame Merkmale. In Tabelle 2.2 gibt es die Klassen „yes“ und „no“. Je nach den Attributwerten der Beispiele geht eine Person Golf spielen (Klasse = yes) oder sie geht nicht Golf spielen (Klasse = no). Es werden immer solange Regeln für eine Klasse gesucht, bis gewisse Kriterien erfüllt sind, dann werden für die nächste Klasse Regeln gelernt. Es kann beliebig viele Klassen geben. Es gibt Zweiklassenprobleme in denen nur zwei Klassen gelernt werden und Mehrklassenprobleme. Bei Mehrklassenproblemen wird für jede Klasse eine Theorie gelernt. Ein Beispiel hierfür stellen One-against-All Verfahren dar.

Definition 2.7 (Regel) *Eine Regel besteht zumeist aus zwei Teilen:*
Einem Regelkörper und einem Regelkopf. Die Regel besteht aus dem Regelkopf, der die Klassenzuweisung enthält. Diese tritt in Kraft, wenn die ***Bedingungen*** *des Regelkörpers erfüllt sind.*

Zur Verdeutlichung wieder ein Beispiel für die Klasse *Golf spielen*:

```
play = yes := windy = false & temperature = cool
```

Diese Regel klassifiziert ein Beispiel als „yes“, wenn die Bedingungen in der Regel erfüllt sind. Diese wären hier windy=false und temperature=cool. Die Regel besteht aus dem Regelkopf (play = yes), der die Klassenzuweisung enthält, die in Kraft tritt, wenn die Bedingungen des Regelkörpers erfüllt sind. Der Regelkörper besteht aus zwei Bedingungen mit jeweils dem Komparator =.

Um diese Regeln zu lernen, wird mit der *Trainingsmenge* gearbeitet. Eine Regel sollte natürlich möglichst viele Beispiele abdecken und richtig klassifizieren. Es gibt zwei „Arten“ zu einer Regel zu kommen: Im induktiven **Top-Down-Ansatz** wird mit einer leeren Regel begonnen, die alle Beispiele abdeckt, dann wird diese Regel durch hinzufügen von Bedingungen verfeinert, bis gewisse Kriterien erfüllt sind. Zusätzlich gibt es noch den **Bottom-Up-Ansatz**, der mit allen möglichen Bedingungen anfängt und dann die Regel verfeinert. Dies wird durch das Entfernen von Bedingungen erreicht. In dieser Diplomarbeit werden ausschließlich Top-Down Ansätze betrachtet.
Welche Bedingungen zum hinzufügen oder herausstreichen ausgewählt werden, wird durch so genannte **Heuristiken**, welche später vorgestellt werden, gesteuert. Normalerweise werden mit einer Regel nicht alle positiven Beispiele der Trainingsmenge abgedeckt. Deswegen kommt häufig ein so genannter **Separate and Conquer** Ansatz zum Einsatz.

Definition 2.8 (Bedingungen oder Attributtest) *Ein Attributtest ist eine Bedingung, die fordert, dass ein Attribut A einen bestimmten Wert $x \in \mathcal{D}_\mathcal{A}$ hat, oder dessen Wert x auf einem Wertebereich W mit $W \subseteq \mathcal{D}_\mathcal{A}$ beschränkt ist, also $x \in W$ gilt.*

Um Bedingungen zu formulieren, werden **Komparatoren** benötigt.

Definition 2.9 (Komparator) *Es werden die Komparatoren $=$ und $\neq$ für nominale Attribute und $\leq$ bzw $>$ für numerische Attribute verwendet. Eine Bedingung gilt als erfüllt, wenn*

- *Fall $=$: der Attributwert dem angegebenen Wert entspricht,*
- *Fall $\neq$: der Attributwert und der angegebene Wert verschieden sind,*
- *Fall $\leq$: der Attributwert kleiner oder gleich dem angegebenen Wert ist,*
- *Fall $>$: der Attributwert größer als der angegebene Wert ist.*

Definition 2.10 (Kandidat) *Ein Kandidat ist eine Regel, die noch evaluiert wird, um zu prüfen, ob sie für eine weitere Verwendung interessant ist.*

Definition 2.11 (Klassifizierung) *Als Klassifizierung wird der Vorgang der Einteilung von Beispielen in Klassen bezeichnet [25].*

Anmerkung: Die Verwendung des Begriffs *Klassifizierung* ist nicht einheitlich.

Definition 2.12 (Positiv-Beispiel) *Um ein Positiv-Beispiel handelt es sich, wenn die Klasse des Beispiels mit der gesuchten Klasse übereinstimmt.*

Ein Beispiel für ein Positiv-Beispiel der Klasse Play=yes aus Tabelle 2.2:

outlook = rainy, temperature = cool, humidity = normal,windy = FALSE, play* = yes
*= zum Lernen steht diese Information nicht zur Verfügung.

Definition 2.13 (Negativ-Beispiel) *Um ein Negativ-Beispiel handelt es sich, wenn die Klasse des Beispiels mit der gesuchten Klasse* ***nicht*** *übereinstimmt.*

Ein Beispiel für ein Negativ-Beispiel der Klasse Play=yes aus Tabelle 2.2:

outlook = sunny, temperature = hot, humidity = high, windy = FALSE, play* = no
*= zum Lernen steht diese Information nicht zur Verfügung.
Aus diesen Beispielen werden (durch später vorgestellte Verfahren), Regeln gelernt. Um neuen Daten, deren „Lernbegriff“ man noch nicht kennt, zu klassifizieren, werden diese Regeln angewendet. Dabei hilft eine Regel in dem Sinne, dass sie sagen kann: „Dieses Beispiel gehört zur Klasse X, da die Bedingungen A und B und C erfüllt sind“.

Definition 2.14 (Ausdruck) *Bei einem Ausdruck handelt es sich um eine Bedingung, eine Regel oder eine Regelmenge.*

Definition 2.15 (Universelle Regel) *Die universelle Regel besteht aus einer Vereinigung von leeren Attributtests und deckt somit alle Beispiele ab.*

Definition 2.16 (Entfernung / Abstand) *Die Entfernung, auch Abstand genannt, wird definiert als die Anzahl der unterschiedlichen Attribute zweier Beispiele. Eine Entfernung von 0 bedeutet, dass die Beispiele exakt gleich sind, während eine Entfernung von n bedeutet, dass sich genau n Attribute bei beiden Beispielen unterscheiden.*

Für numerische Werte gibt es in der Implementierung eine Besonderheit: Die Entfernung zwischen zwei numerischen Werten kann maximal 1 sein, da nur darauf geachtet wird, ob sich zwei Werte unterscheiden. Nicht aber wie groß dieser Unterschied ist.

Definition 2.17 (Extensions) *Extensions sind Attribute, die es erlauben, zwischen zwei Beispielen zu unterscheiden. Meist werden Extensions generiert, um ein Beispiel abzudecken und gleichzeitig ein anderes auszuschließen.*

In Tabelle 2.3 sind zwei Beispiele für Extensions angegeben.

Tabelle 2.3: Extension Beispiele

Beispiel	Outlook	temperature	humidity	windy	class
Beispiel 1	sunny	hot	high	false	no
Beispiel 2	overcast	hot	high	false	yes
Extension(s)	sunny	-	-	-	-
Beispiel a	sunny	hot	high	false	no
Beispiel b	sunny	cool	normal	true	yes
Extension(s)	-	hot	high	normal	false

Definition 2.18 (Theorie) *Theorie bezeichnet die Menge der Regeln, die als Ergebnis eines Lernverfahrens erzeugt wird.*

Definition 2.19 (Beam-Search) *Von einer Beam-Search spricht man, wenn nicht alle Regeln, die ein Algorithmus generiert, als Verfeinerungskandidaten behandelt werden, sondern nur die n-besten. Hierbei steht n für die Beamwidth. Die zurückgegebenen Regeln werden Beam genannt und schränken den Suchraum ein.*

Abbildung 2.2: Entstehen einer Theorie

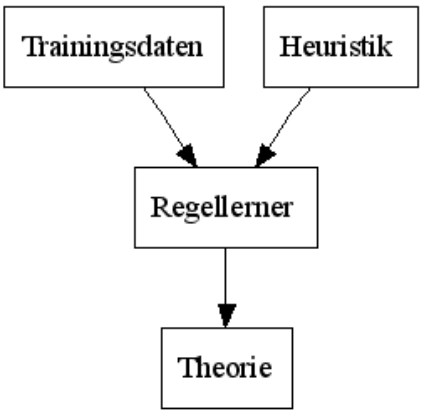
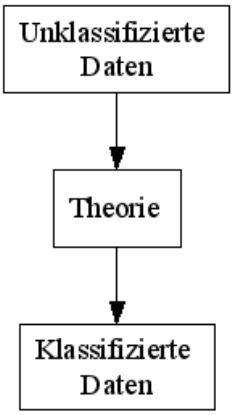


Abbildung 2.3: Klassifikation von neuen Daten



2.2 Der Separate and Conquer (SeCo) Ansatz

SeCo ist die Abkürzung für die Klasse der Separate-and-Conquer Algorithmen. Charakteristisch für diese Algorithmen ist, dass sie nach einer Regel suchen, die einen Teil der Trainingsdaten erklärt, also abdeckt (Conquer), und dann die abgedeckten Beispiele aus den Trainingsdaten entfernt (Separate). Dies geschieht solange bis entweder keine positiven Beispiele in der Trainingsmenge mehr vorhanden oder bis andere, fest definierte, Kriterien erfüllt sind (siehe auch [2]). So soll durch den Algorithmus sichergestellt werden, dass jedes positive Beispiel auch durch mindestens eine Regel abgedeckt wird. Die abstrakte Form, der **Kernalgorithmus**, wird in Abschnitt 2.5 behandelt.

Die Merkmale, in denen sich die SeCo-Algorithmen unterscheiden, lassen sich in drei Gruppen unterteilen:

Hypothesensprache
Die Hypothesensprache begrenzt den Suchraum für Hypothesen, die der Algorithmus lernen kann, denn sie ist die Beschreibungssprache für den zu erlernenden Begriff. Die Hypothesensprache setzt sich aus Regeln zusammen. Wenn in einer Hypothesensprache z.B. nur der Komparator $=$ für nominale Attribute vorkommt, dann muss der Algorithmus nicht mehr die möglichen Kombinationen von Attributen mit $\neq$ berücksichtigen, was den Suchraum stark einschränkt (halbiert).

Suchverfahren
Das Suchverfahren soll neue Hypothesen finden. Es besteht aus Suchalgorithmus, Suchstrategie und Suchheuristik. Die Aufgabe des *Suchalgorithmus* ist es, neue Hypothesen innerhalb des Suchraums zu finden. Es wird mit einer Ausgangshypothese begonnen und verfeinert. Das Bestimmen dieser Ausgangshypothese ist von der Suchstrategie abhängig. Als Ergebnis dieser Verfeinerung erhält man vie-

le Kandidaten, aus denen man die Besten auswählen muss. Aus Speicher- und Laufzeitgründen kann man nicht alle Kandidaten berücksichtigen. Das Filtern der Kandidaten übernimmt die *Suchheuristik*, die meist mit statischen Verfahren versucht, diesen bestimmte Bewertungen zuzuordnen, so dass die *besten* Kandidaten die höchste Bewertung erhalten. Was die besten Kandidaten sind, wird mit Hilfe von Heuristiken und vorher festgelegten Bedingungen bestimmt. Die *Suchstrategie* gibt unter anderem die Richtlinien vor, nach denen Kandidaten verfeinert werden (mehr dazu in Abschnitt 3.1.2 und 3.2.3).

Heuristik zur Vermeidung von Überbestimmtheit wird auch Overfitting genannt. Wenn sich die Hypothese zu sehr an einzelne Beispiele der Trainingsmenge anpasst, dann spricht man von Überbestimmtheit oder auch **Overfitting**. Einerseits ist eine überbestimmte Hypothese wenig interessant, da sie meist nur auf wenige Beispiele anwendbar (und daher für die Abdeckung von anderen Beispielen nicht ausreichend generalisiert) ist. Andererseits besteht eine hohe Wahrscheinlichkeit, dass die Hypothese falsch ist. Dies liegt daran, dass einzelne Beispiele, auf denen die Hypothese beruht, falsch sein können. Viele SeCo-Algorithmen verwenden daher eine Heuristik, die entweder das Erzeugen solcher Hypothesen von vornherein verhindern, oder nachträglich solche Hypothesen aus der Theorie entfernen soll.

Im Allgemeinen werden einfache Regeln, die viele Beispiele abdecken, denen vorgezogen, die komplex sind bzw. nur wenige Beispiele abdecken.

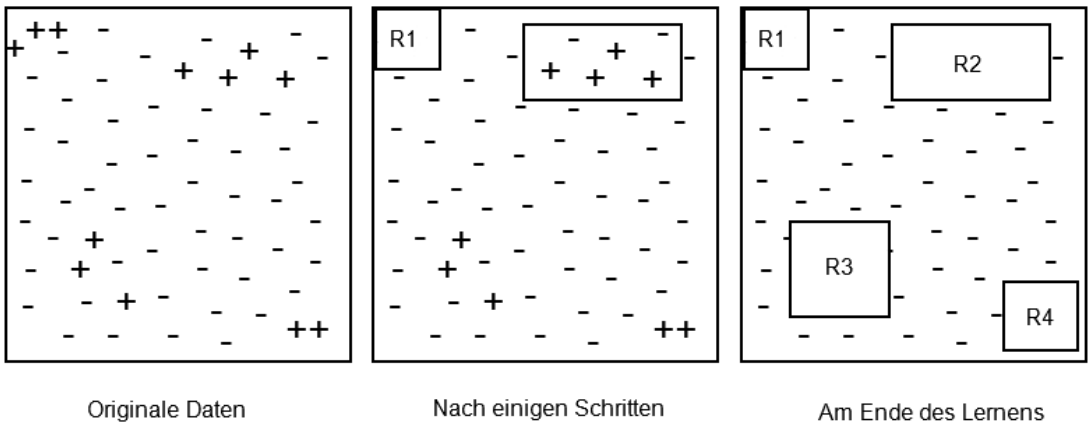


Abbildung 2.4: Generelles Vorgehen eines SeCo-Algorithmus [18]

2.3 Familien von SeCo-Algorithmen

Nach [1] ist das Ergebnis jedes SeCo-Algorithmus eine Beschreibung des zu erlernenden Konzepts. Diese Beschreibung kann sehr unterschiedlich dargestellt wer-

den und ist abhängig von der Hypothesensprache. Durch die Darstellungsform von Konzeptbeschreibungen lassen sich drei verschiedene Familien von SeCo-Algorithmen unterscheiden. Die erste Familie arbeitet mit einer DNF Beschreibung, die zweite mit einer KNF Beschreibung und die dritte mit einer Mischung von beiden.

2.3.1 Die DNF-Familie

Eine durchaus weit verbreitete Form der Konzeptbeschreibung ist die disjunktive Normalform, auch DNF genannt. Diese wird in dieser Diplomarbeit genutzt. Hierbei wird für jede Klasse eine Disjunktion von Konjunktionen erzeugt, die von den zugehörigen Beispielen erfüllt sein mussß. Eine DNF setzt sich aus einzelnen Literalen zusammen und hat die folgende Form:

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

Als Literale werden Attributvergleiche wie outlook = sunny eingesetzt. Vertreter dieser Familie sind die Bexa-Algorithmen (siehe [10]) und die AQ-Algorithmen, die in Abschnitt 3.1 beschrieben werden.

2.3.2 Die KNF-Familie

KNF steht für Konjunktive Normalform, im englischen eher bekannt als CNF, Conjuntive Normal Form, was einer Konjunktion von Disjunktionen entspricht. Die KNF läßt sich durch Negation auch aus der DNF herleiten. Wenn wir das Beispiel aus der DNF nehmen und definieren, dass $\neg L_{i,j}$ die Negation von $L_{i,j}$ ist, dann sieht dieses Beispiel in KNF wie folgt aus:

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

Ein Algorithmus für diese Form wird in [9] beschrieben.

2.3.3 Entscheidungslisten

Entscheidungslisten bestehen aus Einträgen, die jeweils eine Verknüpfung von Bedingungen und eine Klasse beinhalten. In einer Entscheidungsliste können auch verschiedene Klassen vorkommen. Eine Klasse ist hier die Einordnung eines Beispiels. Jeder Eintrag in einer Entscheidungsliste formuliert also eine Regel wie

$$L_1 \wedge L_2 \wedge L_3 \rightarrow class(I) = c$$

mit I als Instanz und $c \in C$ (C ist die Menge aller Klassen). Soll nun geprüft werden, zu welcher Klasse ein Beispiel I gehört, dann werden alle Regeln der Liste der Reihe nach auf I angewendet. Wenn eine Regel erfüllt ist, hat man die Klasse des Beispiels I gefunden. Die nachfolgenden Regeln werden nicht berücksichtigt.

Am Ende der Liste gibt es eine **Default-Regel**, die dann greift, wenn keine der anderen Regeln erfüllt werden konnte. Meistens wird als Default-Regel die Zuweisung zur am häufigsten vorkommenden Klasse c vorgenommen.
Entscheidungslisten können sowohl auch in DNF formulieren, genauso kann man DNF auch in Entscheidungslisten konvertieren. Es gilt aber bei der Konvertierung von Entscheidungslisten zu DNF einige Dinge zu beachten. Die Regeln der Entscheidungsliste dürfen nicht einfach alle mit einem $\wedge$ verbunden werden, da jede Regel für sich keine allgemein gültige Aussage darstellt. Vielmehr basiert die Regel darauf, dass alle vorherigen Regeln nicht zutreffen und somit nur noch wenige andere Bedingungen getestet werden müssen. Dies erlaubt einerseits sehr kurze Regeln, erschwert aber andererseits die Konvertierung in reine DNF Form.
Populäre Algorithmen, die auf Entscheidungslisten basieren, sind die ordered CN2 Regellerner. Diese werden in Abschnitt 3.2 genauer beschrieben. Bei den CN2-Algorithmen unterscheidet man zwei Varianten: die ordered Varianten nutzen Entscheidungslisten, während bei den unordered Varianten ein One-against-All Vorgehen gewählt wird. Der Unterschied zwischen einem Zweiklassenproblem und einem Mehrklassenproblem ist folgender: Bei einem Zweiklassenproblem wird meist ein Konzept gelernt; man hat Beispiele, die zu dem Konzept gehören (positive Beispiele) und Beispiele, die nicht zu dem Konzept passen (negative Beispiele). Ein Mehrklassenproblem mit n-Klassen wird hier in n Zweiklassenprobleme aufgeteilt. Dieses wird auch One-against-All Lernen genannt.
In der Grafik 2.5 ist ein Beispiel mit drei Klassen (a,b,c) dargestellt. Im ersten Schritt werden alle Beispiele der Klasse a als positive Beispiele behandelt und die Beispiele der anderen Klassen als negative Beispiele. Nachdem eine Theorie für a gelernt wurde, werden die Beispiele von b als positive Beispiele behandelt und die Beispiele der Klassen a und c als negative Beispiele. Im dritten und letzten Schritt werden dann die Beispiele der Klasse c als positive Beispiele behandelt und die Beispiele der Klassen b und c als negative Beispiele. Am Ende des Lernprozesses hat man drei Theorien. Für jede Klasse existiert eine Theorie.

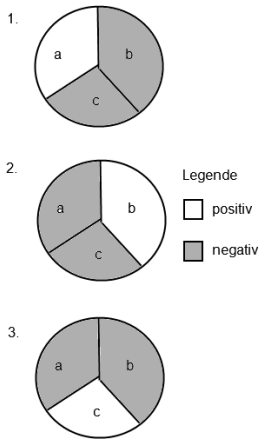


Abbildung 2.5: Lernen eines Multiklassenproblems

2.4 Vergleichen von SeCo-Algorithmen

Um die verschiedenen Varianten der SeCo-Algorithmen vergleichen zu können, werden die Varianten auf dieselben Lernprobleme angewandt. Danach werden die Ergebnisse verglichen. Die bei den Ergebnissen relevanten Aspekte werden in den folgenden Abschnitten erläutert.

2.4.1 Korrektheit

Die Korrektheit sagt aus, wie viel Prozent der zu klassifizierenden Beispiele korrekt klassifiziert wurden. Dabei kann sich Korrektheit auf die Trainings- sowie auf die Testmenge beziehen. Betrachtet man die Korrektheit auf der Trainingsmenge, so wird die gelernte Theorie auf die Trainingsmenge angewandt. Es kann immer eine Genauigkeit von bis zu 100% erreicht werden, es sei denn, die Trainingsdaten sind widersprüchlich. Im Allgemeinen haben Regellerner auf den Trainingsdaten eine geringere Genauigkeit als 100%. Deswegen ist dieser Wert meistens eher uninteressant. Im anderen Fall wird die gelernte Theorie auf die Testmenge, die dem Lern-Algorithmus nicht zur Verfügung stand, angewendet. Um dafür nicht dem Lerner Beispiele vorzuenthalten, wendet man eine *10-fold cross validation* an (siehe [13]). Dabei wird die gegebene Beispielmenge vor Lernbeginn in 10 gleich große Teile zerlegt. Der Lerner bekommt zum Lernen 9 dieser Teile, der letzte Teil wird zum Testen benutzt. Dies wird mit allen anderen Teilen wiederholt, bis jeder der 10 Teile einmal Testmenge war. Die daraus gewonnenen Korrektheiten werden arithmetisch gemittelt und als Ergebnis für den Lerner, der alle Beispiele zur Verfügung hat, verwendet. Dies kann natürlich nur eine Abschätzung der Korrektheit des Lerners sein. Die Korrektheit gibt also eine Schätzung für die Wahrscheinlichkeit an, mit der die Prognosen, die die gelernte Theorie trifft, auf neuen unbekannten Beispielen richtig ist.

2.4.2 Größe der Regelmenge

Als weiteres Merkmal der Qualität eines SeCo-Algorithmus wird die Anzahl an Regeln betrachtet, die ein Algorithmus benötigt, um alle Beispiele der Trainingsmenge abzudecken. Obwohl eine große Regelmenge nicht der Korrektheit schaden muss, ist eine kleinere Regelmenge vorzuziehen, da diese leichter zu verstehen ist und seltener zur Überbestimmtheit neigt. Dabei muss aber nicht nur die Anzahl der Regeln, sondern auch die Anzahl der Bedingungen betrachtet werden. Auch darf die Hypothesensprache bei diesen Vergleichen nicht vernachlässigt werden. Eine mächtigere Hypothesensprache benötigt oft weniger Regeln als eine weniger mächtige Hypothesensprache.

2.4.3 Algorithmische Komplexität

Die Komplexität eines Algorithmus gibt dessen Laufzeitverhalten an. Je komplexer ein Algorithmus ist, desto schlechter ist das Laufzeitverhalten und bei größeren Datenmengen steigt die Rechenzeit weiter an. Oft geht eine Verbesserung der Algorithmen mit einer Senkung der Komplexität einher.

2.5 Die Grundstruktur eines SeCo-Algorithmus

Die Gemeinsamkeiten von verschiedenen Separate-and-Conquer Algorithmen sowie die Grundstrukur wurden in [2] zusammengefasst und formuliert. Diese Grundstruktur wird als Kernalgortihmus bezeichnet, der sich in allen SeCo-Algorithmen finden läßt.

Quelltext 2.1 Allgemeiner SeCo-Algorithmus Teil1

```
procedure SeparateAndConquer(examples){
  theory := ∅
  while positive(examples) ≠ ∅
  {
    rule := FindBestRule(examples)
    covered := cover(rule, examples)
    if RuleStoppingCriterion(theory, rule, examples)
      exit while
    examples := examples \ covered
    theory := theory ∪  rule
  }
  theory := PostProcess(theory)
  return (theory)
}
```

Quelltext 2.2 Allgemeiner SeCo-Algorithmus Teil2

```
procedure FindBestRule(examples)
{
  initRule := InitializeRule(examples)
  initVal := EvaluateRule(initRule)
  bestRule := <initVal, initRule>
  rules := { bestRule}
  whiles rules ≠ ∅
  {
    candidates := InitializeRule(examples)
    rules := rules \ candidates
    for candidate ∈  candidates
    {
      refinements := RefineRule(candidate, examples)
      for refinement ∈  refinements
      {
        evaluation := EvaluateRule(refinement, examples)
        while (!StoppingCriterion(refinement,evaluation,examples)
        {
         newRule := <evaluation, refinement>
         rules := InsertSort(newRule, rules)
         if (newRule > bestRule )
           bestRule := newRule
        }
      }
    }
    rules := FilterRules(rules, examples)
  }
  return (bestRule)
}
```

Wie in Quelltext 2.1 und 2.2 zu erkennen ist, gibt es zunächst zwei Prozeduren: SeparateAndConquer und FindBestRule. SeperateAndConquer besteht aus einer Schleife, die versucht, die Theorie für die zu lernende Klasse zu finden. Es wird mit einer leeren Theorie begonnen. Daraufhin wird wiederholt eine „beste Regel“ ermittelt. Solange das RulestoppingCriterion nicht zutrifft, wird die ermittelte Regel der Theorie hinzugefügt und die abgedeckten Beispiele werden aus der Trainigsmenge entfernt. Abhängig vom benutzten Algorithmus, werden nur die positiven Beispiele oder alle abgedeckte Beispiele entfernt. Daraufhin wird PostProcess ausgeführt. Hierbei handelt es sich um eine Nachbehandlung, bei der

u.a. ein Post Pruning durchgeführt wird.
FindBestRule beginnt mit einer initialen Regel, die gleichzeitig als beste Regel gespeichert wird und in die Regelmenge *Rules* aufgenommen wird. Solange in *Rules* noch Regeln enthalten sind, werden aus diesen mit SelectCandidates Kandidaten für die Verfeinerung ausgewählt und der Menge der Kandidaten (*candidates*) hinzugefügt. Für jeden der Kandidaten wird eine Verfeinerung mit Hilfe von RefineRule durchgeführt. Die Varianten des Verfeinerungsprozesses kann man anhand der **Suchstrategie** unterteilen:

- **Top-Down-Strategie**
  Ausgehend von einer allgemeinen Regel möchte man diese Regel spezialisieren. Die allgemeine Regel hat meist die Form: true → Klasse. Die Spezialisierung deckt aber nur eine Teilmenge der Beispiele ab, die die ursprüngliche Regel abdeckte. Also versucht man die hinzugefügte(n) Bedingung(en) so zu wählen, dass möglichst viele Negativ-Beispiele ausgeschlossen werden und gleichzeitig möglichst viele Positiv-Beispiele erhalten bleiben.

- **Bottom-Up-Strategie**
  Ausgehend von einer speziellen Regel (meist einer Regel, die genau ein positives Beispiel abdeckt), wird diese durch das Entfernen von Bedingungen verallgemeinert. Auch bei diesem Schritt möchte man, dass viele Positiv-Beispiele erhalten bleiben und man nur sehr wenige bzw. gar keine Negativ-Beispiele abdeckt.

Die verfeinerten Regeln werden daraufhin mit einer Evaluierungsfunktion (*EvaluateRule*) bewertet und anschließend mit Hilfe des *StoppingCriterion* überprüft. Hierbei soll festgestellt werden, ob eine weitere Verfeinerung dieser Regel sinnvoll ist. Wie dieses Kriterium gewählt werden soll, wird hier nicht vertieft werden.
Jede Verfeinerung, auf die das StoppingCriterion nicht zutrifft, wird in die Regelmenge *Rules* übernommen. Wenn diese Regel besser als die bisher beste Regel ist, wird die beste Regel durch die aktuelle Regel ersetzt. Am Ende der Prozedur wird dann die beste Regel zurückgegeben.

3 Die Algorithmen

In diesem Kapitel wird auf die beiden Algorithmen AQ (siehe [3]) und CN2 (siehe [3] und [4]) vorgestellt sowie auf ihre Besonderheiten, ihre Stärken und ihre Heuristiken eingegangen. Der AQ Algorithmus ist ein älterer Ansatz und der CN2 ein Ansatz neueren Datums.

3.1 Der AQ-Algorithmus

Der hier verwendete **AQR**-Algorithmus (wie in [3]) basiert auf dem **AQ**-Algorithmus (siehe [14]), wobei der AQR eine Rekonstruktion eines offenen AQ-basierten Systems ist. In dieser Arbeit werden AQR und AQ als Synonym füreinander genutzt. Der AQ bildet ein Set aus Entscheidungsregeln, ein Set für jede zu lernende Klasse. Der AQR erlaubt Tests aus den folgenden Komparatoren (siehe Definition 2.6): $\{=, \leq, >, \neq\}$. Ein *Ausdruck* deckt ein Beispiel ab, wenn der Ausdruck für dieses Beispiel *wahr* ist. Die leere Regel deckt alle Bespiele ab. Mit AQ wird ein neues Beispiel klassifiziert, indem geprüft wird, welche der Regeln aus der Regelmenge auf das Beispiel zutreffen. Wenn solch eine Regel gefunden wurde, wird das Beispiel mit der Klassifikation versehen, die der Regel entspricht. Wenn mehr als eine Regel zutrifft, dann wird das Beispiel nach der Häufigkeit der vorkommenden Klassifikation der zutreffenden Regeln klassifiziert. Falls keine Regel zutrifft, bekommt das Beispiel die Klasse der am häufigsten vorkommenden Klasse in der Trainingsmenge (Default-Regel).

3.1.1 Der Lern-Algorithmus des AQR

Der AQR generiert pro Durchgang eine Entscheidungsregel für jede Klasse. Sobald fest steht, auf welche Klasse man sich fokussiert, wird eine Regelmenge für diese Klasse geschaffen. Dies geschieht in Stufen. Jede Stufe generiert eine Regel und entfernt die abgedeckten Beispiele aus der Trainingsmenge. Das wird solange wiederholt, bis alle positiven Beispiele einer Klasse mit Regeln abgedeckt sind. Dieser beschriebene Prozess wird für jede zu lernende Klasse gestartet.

Definition 3.1 (Stern) *Ein Stern (auch Star genannt) enthält eine Menge von Verfeinerungen einer Regel.*

Wenn man die Menge an Regeln in einem Star beschränkt, wird aus einem Stern ein Beam in der Beam-Search.

3.1.2 Heuristiken des AQR-Algorithmus

Die verwendeten Heuristiken der AQ-Algorithmen sind, je nach Implementierung, verschieden. Die Heuristik, die der AQR nutzt, um die beste Regel zu finden, zeichnet sich dadurch aus, dass sie simpel aber effektiv ist. Sie lautet: „Maximiere

die Anzahl der abgedeckten Positiv-Beispiele“. Die Heuristik, um einen Stern zu kürzen, ist „Maximiere die Summe der abgedeckten Positiv-Beispiele und minimiere die Summe der abgedeckten Negativ-Beispiele“. Falls es zu keiner klaren Entscheidung der beiden Heuristiken kommt (dies bezeichnet man auch als **Tie**), wird die Regel bevorzugt, die weniger Bedingungen hat.

3.1.3 Das Seed

Der AQ-Algorithmus bedient sich bei der Verfeinerung seiner Regeln eines so genannten *Seed*. Dieses Seed dient als Bezugspunkt für neue Regeln. Meist wird aus der Menge der positiven Beispiele ein Beispiel zufällig ausgewählt. Dieses zufällig gewählte Beispiel dient der Regelgenerierung und Verfeinerung als Seed. Bei der Regelgenerierung wird als Ausgangs-Regel die leere Regel genutzt. Diese Regel deckt automatisch das Seed ab, da die leere Regel alle Beispiele abdeckt. Abhängig von dem Seed-Beispiel werden nun negative Beispiele gesucht, die auch durch die aktuell zu lernende/verfeinernde Regel abgedeckt werden. Dabei wird auf die Entfernung der Beispiele zu dem Seed-Beispiel geachtet. Sinn und Zweck des Seeds und des „nahen“ Negativ-Beispiels ist folgender: Es wird geprüft, welche Attribute benötigt werden, um das positive Seed-Beispiel, nicht aber das negative Beispiel abzudecken. Diese Attribute werden dann benutzt, um die vorhandene Kandidaten-Regel zu verfeinern. Falls der Fall eintritt, dass es keine Attribute gibt, die zwischen Seed und gewähltem Negativ-Beispiel unterscheiden lassen, wird das Negativ-Beispiel ignoriert und es wird ein anderes Beispiel ausgewählt.

Quelltext 3.1 Der Pseudo-Code des AQR-Algorithmus (nach [3])

```
procedure AQR(pos, neg)
{
  cover := ∅;
  while (!cover.Covers(pos))
  {
    seed := FindSeed(pos, cover);
    newstar := Star(seed, neg);
    best := userHeuristik(newstar);
    cover := cover ∪ best;
  }
  return cover
}

procedure Star(seed, neg)
{
  nstar := Emptyrule;
  while(nstar.covers(neg))
  {
    e_neg := negativeCovered(nstar);
    extension := selectorsCoverAAndNotB(star, e_neg);
    for (newrule ∈ nstar)
    {
      for(newext ∈ extension)
      {
        tempstar := tempstar ∪ (newrule ∧ newext);
      }
    }
    nstar := trimmStarUntilMaximum(tempstar, maximum);
  }
  return nstar;
}
```

Bei dem Vergleich von 3.1 mit 2.1 und 2.2, stellen man fest, dass die Procedure AQR stark der Procedure SeparateAndConquer ähnelt. In beiden Prozeduren werden die abgedeckten Beispiele aus der Menge der Beispiele entfernt und dies solange wiederholt, bis keine positiven Beispiele mehr vorhanden sind. Das Entfernen von Beispielen aus der Trainingsmenge ist charakteristisch für Separate-and-Conquer Algorithmen. Somit muss der dargestellte AQR auch ein Separate-and-Conquer Algorithmus sein.

Im Pseudo-Code ist viel Spielraum für Interpretation gegeben. Daher wird auf die genaue Realisierung des Algorithmus aus Quelltext 3.1 erst in Kapitel 4.3.1 eingegangen.

3.2 Der CN2-Algorithmus

Es gibt zwei Varianten des CN2-Algorithmus: eine „ordered"-Variante, die Entscheidungslisten lernt und eine „unordered"-Variante, die einen One-against-All Lerner darstellt. Der CN2-Algorithmus, der in dieses Arbeit verwendet wird, ist eine „ordered"-Variante. Dieser produziert geordnete „wenn-dann"-Regeln, während auf AQ basierende Algorithmen ungeordnete „wenn-dann"-Regeln generieren. Dabei hält der CN2 an der „*Beam-Search*"-Methode des AQ-Algorithmus fest. Der CN2 ist jedoch nicht mehr abhängig von speziellen Beispielen. Auch schließt der CN2 Regeln in die Suche mit ein, welche die Trainingsdaten nicht perfekt erklären. Durch das Durchführen einer Top-Down-Suche ist es mit CN2 nun möglich, das Lernen und Verfeinern von weiteren Regeln zu stoppen, sobald diese Regeln eine geringe statistische Relevanz haben.
Nachfolgend werden vier Dimensionen des Algorithmus erklärt: Die Hypothesensprache, das Suchverfahren, die Heuristik und den Pseudo-Code.

3.2.1 Hypothesensprache des CN2

Der CN2-Algorithmus benutzt Regeln in geordneten Listen, Regeln die nicht aufeinander aufbauen, aber in denen alle Regeln für eine Klasse zusammenstehen. Die Form der einzelnen Regeln entspricht denen des AQ-Algorithmus. Also

`if <Ausdruck> then Vorhersage=<Klasse>`.

Der CN2-Algorithmus hat am Ende seiner gelernten Regeln immer eine „Default-Regel". Diese sagt die Klasse vorher, die in den Trainings-Beispielen am häufigsten vorkam. Der Sinn der Default-Regel liegt darin, dass ein Beispiel auf jeden Fall klassifiziert wird, auch wenn eigentlich keine Regel bei dem Beispiel feuert. Um ein neues Beispiel zu klassifizieren, wird nacheinander so lange jede Regel ausprobiert, bis das Beispiel einer Klasse zugeordnet werden kann. Die Klassenvorhersage der Regel wird also dem noch unklassifizierten Beispiel zugeordnet. Wenn keine gelernte Regel von dem Beispiel erfüllt werden kann, greift die Default-Regel.

3.2.2 Der Lern-Algorithmus des CN2

Der Pseudocode des CN2 findet sich in Quelltext 3.2. Der CN2-Algorithmus ist ein iterativer Algorithmus. In jeder Iteration sucht der Algorithmus nach einem Ausdruck, der eine große Menge an Beispielen einer Klasse C und nur wenige von einer anderen Klasse abdeckt. Der Ausdruck sollte zuverlässige Vorhersagen im Sinne der CN2 Evaluierungsfunktionen treffen. Die durch den Ausdruck abgedeckten Beispiele werden aus der Trainingsmenge entfernt und die Regel

`if Ausdruck then Vorhersage = C` am Ende der Regelliste angefügt. Dies wird so lange wiederholt, bis kein neuer Ausdruck gefunden werden kann, der die Heuristiken erfüllt.
Der Algorithmus sucht nach Ausdrücken, indem er eine Top-Down-Suche mit Pruning startet. Auch der CN2 hat wie der AQ-Algorithmus ein Limit (einen so genannten *Star*) aus den bisherigen besten gefundenen Ausdrücken. Es werden nur Spezialisierungen dieses Sets betrachtet. Ein Ausdruck wird spezialisiert, indem entweder ein neuer Konjunktiver Term ($\wedge$) hinzugefügt, oder ein Disjunktives Element ($\vee$) entfernt wird.
Es wird also immer mit einer Default-Regel der Form
$true \rightarrow class = +$ begonnen. So könnte z.B.: die Verfeinerung einer Hypothese über zwei Iterationen aussehen:

$$\begin{aligned} true &\rightarrow class = + \\ (outlook = sunny) &\rightarrow class = + \\ (outlook = sunny) \wedge (temperature \leq 30) &\rightarrow class = + \end{aligned}$$

Um nun zu überprüfen, welche Bedingungen hinzugefügt werden können, müssen diese erstmal erzeugt werden. Zunächst werden alle möglichen Attributtests erzeugt und für jede Regel alle Kombinationen berechnet, die durch das Hinzufügen eines Attributtests entstehen. Angenommen die Menge aller Regeln in der i-ten Iteration sei R_i und AtT seien alle möglichen Attributtests. Dann werden pro Iteration maximal $|Atributtests| * |R_i|$ neue Regeln erzeugt. Dieses stellt einen sehr großer Rechenaufwand dar. Deswegen werden direkt ein paar Optimierungen eingebunden. Es bietet sich an, Regeln nicht zu erzeugen, deren Bedingungen nicht erfüllt werden können.
Regeln wie $(outlook = sunny) \wedge (outlook = windy) \rightarrow class = +$ können beispielsweise nie erfüllt werden und müssen deswegen auch nicht erzeugt werden. Die übrigen Regeln werden mit Hilfe einer Heuristik bewertet. Daraufhin werden alle „schlechten“ Regeln aus der Menge entfernt.

3.2.3 Heuristiken des CN2

Zum Bewerten von Regeln wird die *LaplacePrecision* Heuristik (siehe [2] oder Definition 3.1) des CN2-Algorithmus verwendet.

Definition 3.2 (LaplacePrecision)

$$LPA(r) = \frac{n_c + 1}{n_{tot} + k}$$

- r, *die zu bewertende Regel*
- n_c, *die Anzahl der positiven*[3] *Beispiele, die von der Regel abgedeckt werden*
- k, *die Anzahl der Klassen*
- n_{tot}, *Summe der Beispiele, die durch die Regel* r *abgedeckt werden*

Die Bewertungsfunktion ordnet nun jeder verfeinerten Regel einen LPA(r)-Wert zu. So können Regeln nach ihrer „Qualität“ geordnet und gute von schlechten Regeln unterschieden werden.

Ein zusätzlicher Test soll verhindern, dass Regeln weiter verfeinert werden, die zu diesem Zeitpunkt bereits zu wenige positv-Beispiele und zu viele Negativ-Beispiele abdecken - der **Signifikanztest**. Der CN2 hat als Signifikanztest meist die *Loglikelihood-Ratio-Statistic* (siehe auch [3]), bei der eine Regel einen bestimmten Grenzwert überschreiten muss, damit die Regel als signifikant betrachtet wird. Die Statistik ist χ^2(Chi2) verteilt und der Schwellwert kann, je nach gewünschtem Signifikanz-Niveau, verändert werden.

[3] Der Klasse c zugehörigen

Definition 3.3 (Loglikelihood-Ratio-Statistic)

$$LSR(r) = 2 \cdot (p \cdot log(\frac{p}{e_p}) + n \cdot log(\frac{n}{e_n}))$$

- *r, die zu bewertende Kandidaten Regel*
- *p, Anzahl der positiven, durch r abgedeckten Beispiele*
- *n, Anzahl der negativen, durch r abgedeckten Beispiele*
- *P, Anzahl der positiven Trainingsbeispiele in der Trainingsmenge*
- *N, Anzahl der negativen Trainingsbeispiele in der Trainingsmenge*

Für e_p und e_n gilt:

$$e_p = (p + n) * \frac{P}{P + N}$$

$$e_n = (p + n) * \frac{N}{P + N}$$

e_p und e_n steht für die Anzahl an erwarteten positiven und negativen Beispielen, sofern davon ausgegangen wird, dass die (durch die Regel) abgedeckten Beispiele p+n im gleichen Verhältnis wie in den Gesamtdaten verteilt sind.

Quelltext 3.2 Der Pseudo-Code des CN2-Algorithmus (nach [3])

```
procedure CN2(E)
{
  rulelist := ∅
  repeat
  {
    bestcpx := FindBestComplex(E);
    if (bestcpx := ∅);
    {
      e' := bestcpx.covers;
      e := e ∩ e';
      c := MostCommonClass(e);
      rulelist := rulelist ∪ (true→c);
    }
  } until (bestcpx == ∅ or e == ∅)
  return rulelist;
}

procedure FindBestComplex(E)
{
  star := EmptyComplex;
  bestcpx := ∅;
  selectors := GetAllSelectors;
  while(star != ∅)
  {
    for(tempcomplex ∈ star)
    {
      for(ext ∈ selectors)
      {
        newcomplex := (tempcomplex ∧ ext);
        if(newcomplex ∉ star & Achievable(newcomplex))
        {
          newstar := newstar ∪ newcomplex;
        }
      }
    }
    for(ci ∈ newstar){
      if(SignificantTest(ci, e) & BetterThanBest(ci))
      {
        bestcpx := ci;
      }
    }
    star := RemoveWorstUntilMax(newstar, maximum);
  }
  return bestcpx;
}
```

4 Algorithmen im SeCo-Framework

In diesem Kapitel wird darauf eingegangen, wie im Allgemeinen ein Algorithmus im SeCo-Framework implementiert wird, worauf bei der Implementierung zu achten ist und welche Möglichkeiten für die Parametrisierbarkeit es in SeCo gibt. Dann wird, am Beispiel des AQ Algorithmus, die Vorgehensweise der Implementierung aufgezeigt. Zum Schluss wird auf die XML-Konfigurationsdatei des Seco-Frameworks eingegangen. Weitere Informationen zu dem SeCo Framework finden sich in [1]

4.1 Wichtige Funktionen

Der allgemeine Code eines SeCo-Algorithmus ist bereits in Quelltext 2.1 vorgestellt worden. Nun werden einige wichtige Funktionen des Kern-Algorithmus definiert. Welche dies sind zeigt Tabelle 4.1, nach [1].

Funktion	Beschreibung
RuleStoppingCriterion	Das Kriterium bestimmt, wann der Lernprozess endet.
PostProcess	Ermöglicht die Nachbearbeitung der Regeln.
InitializeRule	Bestimmt die erste Regel, mit der die Verfeinerung begonnen wird.
EvaluateRule	Bewertet Regeln mit Hilfe einer Heuristik.
SelectCandidates	Bestimmt die nächsten Kandidaten der Regeln, die verfeinert werden.
RefineRule	Verfeinert eine Regel und gibt alle Verfeinerungen zurück.
StoppingCriterion	Entscheidet mit Hilfe einer Heuristik, ob eine weitere Verfeinerung der Regel sinnvoll ist.
FilterRules	Ermöglicht die Menge der Regeln nach gewissen Kriterien zu filtern.

Tabelle 4.1: Variable Funktionen eines SeCo-Algorithmus

Diese Funktionen müssen jetzt genauer definiert werden. Es gibt von fast jeder Funktion eine Standard-Funktion, die so genannte triviale Funktion. Als Beispiel SelectCandidates: Die triviale SelectCandidates Methode gibt *alle* zu verfeinernden Regeln zurück. Wenn jetzt ein Algorithmus in der SeCo-Factory implementiert werden soll, muss überlegt werden, welche Teile des Algorithmus welche Funktion der Factory haben. Diese Teile existieren meist in vorgegebenen Pseudo-Codes. Wie so eine Umwandlung eines Algorithmus aus Pseudo-Code in Funktionen für die SeCo-Factory aussehen kann, wird anhand des AQ-Algorithmus gezeigt. Mehr Einzelheiten zur SeCo-Factory finden sich in [1]. In Abbildung 4.1 wird die Packet-

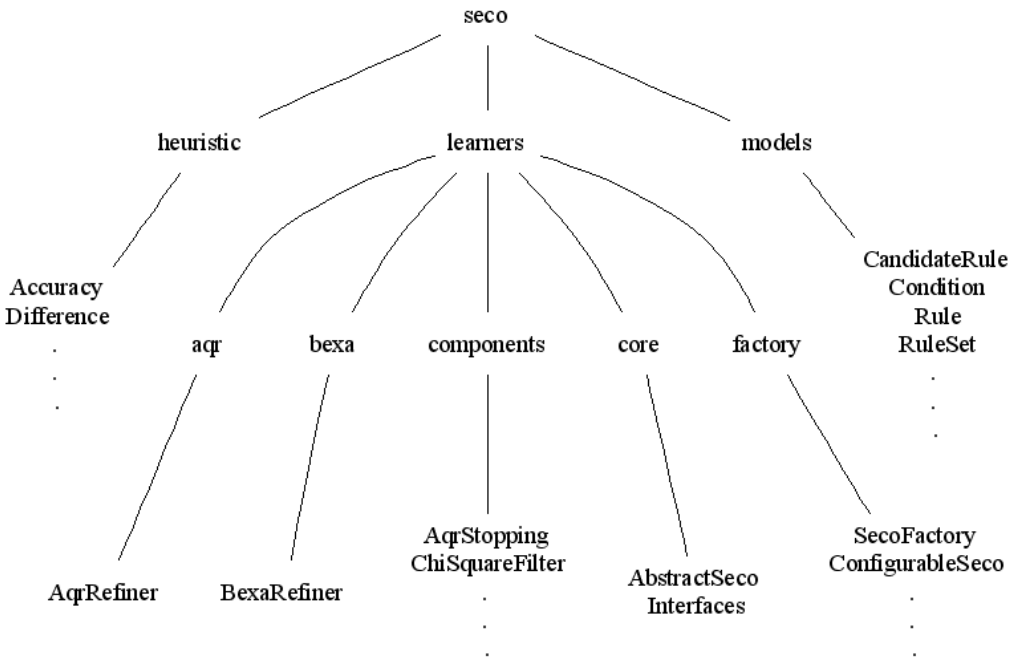


Abbildung 4.1: Die SeCo-Hierarchie

Hierarchie des SeCo-Frameworks dargestellt.

4.2 Die Konfiguration eines SeCo-Algorithmus

4.2.1 Die Komponenten eines SeCo-Algorithmus

In SeCo gibt es folgende Komponenten:

- **CandidateSelector** Wählt die Kandidatenregeln zur Verfeinerung aus,
- **PostProzessor**(Optional) Dient der Nachbearbeitung einer gelernten Theorie,
- **RuleEvaluator** Die Bewertungsfunktion von Regeln,
- **RuleFilter** Kann Regeln aus einer Menge herausfiltern,
- **RuleInitializer** Legt fest wie die erste Regel erzeugt werden soll,
- **RuleRefiner** Erzeugt neue Regeln, indem es andere Regeln modifiziert oder verbessert,
- **RuleStoppingCriterion** Definiert ein Kriterium, welches entscheidet, wann der Lernprozess endet,
- **StoppingCriterion** Definiert ein Kriterium, welches entscheidet, ob eine Regel nochmal verfeinert wird oder nicht.

Dieses sind die „variablen“ Funktionen der SeCo-Factory. Wie und mit welchen Parametern diese benutzt werden sollen, wird der SeCo-Factory über eine XML-Datei mitgeteilt.

4.2.2 Heuristiken eines SeCo-Algorithmus

Johannes Fürnkranz hat eine abstrakte Suchheuristik entworfen, die in dieser Diplomarbeit als Grundlage für die Umsetzung weiterer Heuristiken dient, da sie eine Schnittstelle definiert, die von den verschiedensten Heuristiken benutzt werden kann. Die vorhanden Such-Heuristiken werden in den folgenden Tabellen (Tabelle 4.2 und Tabelle 4.3) dargestellt.

Tabelle 4.2: Variablendefinition für die Heuristiken(aus [1])

Variable	**Beschreibung**
P	Die Gesamtzahl der positiven Beispiele
N	Die Gesamtzahl der negativen Beispiele
r	Die Kandidatenregel
tn	Anzahl der Beispiele, die von r korrekterweise negativ klassifiziert sind (true negatives)
fn	Anzahl der Beispiele, die von r fälschlicherweise negativ klassifiziert sind
tp	Anzahl der Beispiele, die von r korrekterweise positiv klassifiziert sind (true positives)
fp	Anzahl der Beispiele, die von r fälschlicherweise positiv klassifiziert sind

4.2.3 Konfiguration per XML-Datei

Nachdem alle Funktionen implementiert sind, muss der SeCo-Factory mitgeteilt werden, aus welchen Teilen sie einen Lern-Algorithmus zusammenstellen soll. Dies geschieht über eine XML-Datei, die beim Aufruf der Factory übergeben wird. Wie diese XML-Datei aufgebaut ist, wird nachfolgend vorgestellt. In dieser XML-Datei gibt es verschiedene Elemente und Attribute, diese werden in Tabelle 4.5 (entnommen aus [1]) erläutert. Genauere Einzelheiten zu den Eigenschaften der XML-Datei und deren Entsprechungen in dem SeCo-Framework gibt es in [1] Kapitel 4.

Tabelle 4.3: Heuristiken im SeCo Framework (nach [2])

Name der Heuristik	**Formel**
Accuracy	$A(r) = \frac{tp+tn}{P+N}$
Correlation	$Corr(r) = \frac{tp*tn-fn*fp}{\sqrt{P*N*(tp+fp)(fn+tn)}}$
Difference	$Diff(r) = tp - fp$
FoilGain	$Foil(r) = tp * (\log_2 \frac{tp}{tp+fp} - \log_2 c)$
Laplace	$Lap(r) = \frac{tp+1}{tp+fp+2}$
LinearCost	$L(r) = c * tp - (1 - c) * fp$
MEstimate	$M(r) = \frac{tp+m\frac{P}{P+N}}{tp+fp+m}$
Precision	$Prec(r) = \frac{tp}{tp+fp}$
RateDiff	$RD(r) = \frac{tp}{P+N} - \frac{fp}{fp+tn}$
WRAcc	$WRAcc(r) = \frac{tp}{P+N} - \frac{tp+fp}{P+N} * \frac{P}{P+N}$

4.2.4 Default-Komponenten der SeCo

Das SeCo-Framework hat einige Default-Komponenten, die nicht explizit in der XML-Konfiguration angeben werden müssen. Welche das sind, wird in Tabelle 4.4 dargestellt (nach [1]). Diese Komponenten befinden sich alle im *seco.learners.components* package.

Tabelle 4.4: Die Default-Komponenten des SeCo-Frameworks

Name der Komponente	**Beschreibung**
DefaultSelector	Dieser CandidateSelector wählt alle Kandidatenregeln aus, die daraufhin verfeinert werden.
DefaultRuleEvaluator	Hier wird die LaPlace-Heuristik zur Bewertung von Regeln verwendet.
DefaultRuleInitializer	Wählt als initiale Regel eine leere Regel, die keine Bedingungen enthält und im Regelkopf die zu erlernende Klasse enthält.
DefaultRuleStop	Dieses Kriterium prüft, ob eine gegebene Regel besser ist als eine leere Regel, die nur im Regelkopf die aktuell zu erlernende Klasse beinhaltet. Bewertet werden dabei die Regeln mit der LaPlace-Heuristik, sofern nicht eine andere Heuristik angegeben wird.

Tabelle 4.5: Elemente und Attribute der SeCo XML-Beschreibung

Element	**Attribute**	**Beschreibung**
seco		Das Wurzelelement, welches die Konfiguration eines SeCo-Klassifizierers beinhaltet
secomp		Eine SeCo-Komponente
	interface	Die Schnittstelle bzgl. der AbstractSeco-Klasse[3]. Mögliche Werte sind candidateselector, postprocessor, ruleevaluator, rulefilter, ruleinitializer, rulerefiner, rulestoppingcriterion und stoppingcriterion.
	classname	Name der Java-Klasse, welche die Komponente implementiert.
	package	(Optional) Das Java-Package, welches die angegebene Java-Klasse beinhaltet. Als Standardwert wird *seco.learners.components* angenommen.
jobject		Ein beliebiges Java-Objekt
	classname	siehe secomp
	package	siehe secomp
	setter	Teilname der setter-Methode, welche für Aggregation dieses Objekts mit dem Objekt der nächsthöheren Ebene verwendet wird. Wenn also die Java-Methode z.B. setHeuristic heißt, so muß als setter 'heuristic' angegeben werden.
property		Zur Festlegung einer spezifischen Eigenschaft eines Objekts.
	name	Name der Eigenschaft
	value	Wert der Eigenschaft

[3]siehe seco.learners.core.AbstractSeco

4.3 Der AQ-Algorithmus in SeCo

Es wird von dem Pseudo-Code des Kernalgorithmus (Quelltext 2.1) und dem Pseudo-Code des AQ-Algorithmus (Quelltext 3.1) ausgegangen.
Als erstes muss aus dem Code des AQ-Algorithmus alles heraus gefiltert bzw entfernt werden was schon durch den Kern-Algorithmus der SeCo-Factory abgedeckt wird. Darunter fällt das Entfernen der abgedeckten Beispiele, die Schleife, etc..
Es gilt herauszufinden wie der AQ-Algorithmus auf die Funktionen der SeCo-Factory abgebildet werden kann. Es gibt triviale und nicht triviale Funktionen, die für die SeCo realisiert werden müssen.
Beginnen wird mit dem RuleStoppingCriterion. Für den AQ-Algorithmus wird eine eigene Implementierung benötigt, da das Kriterium ist: „Decke alle positiven Beispiele ab und sei besser als die Default-Regel“. Da AQ kein PostProcessing hat, gehört diese Funktion zu den trivialen Funktionen. Die Funktion InitializeRule gehört auch zu den trivialen Funktionen, da AQ auch immer mit der leeren Regel beginnt. EvaluateRule gehört nicht zu den trivialen Funktionen, auch wenn wir eigentlich nur eine andere Bewertungsheuristik übergeben müssen. Wobei die Funktion SelectCandidates zu den trivialen Funktionen gehört, da wir einfach alle Regeln verfeinern wollen. Die aufwändigste Funktion wird die RefineRule. Das StoppingCriterion muss auch für AQ angepasst werden, da das Kriterium folgendes ist „Stoppe das verfeinern einer Regel sobald sie keine negativen Beispiele mehr abdeckt“. Während aber FilterRules wieder eine triviale Funktion wird, da die Menge der Regeln nicht gefiltern wird.
In Quelltext 4.1 und 4.2 finden sich die trivialen und nicht trivialen Funktionen.

Quelltext 4.1 Triviale Funktionen des AQ-Algorithmus

```
procedure PostProcess(theory){
  return theory // optional, von AQ nicht genutzt
}

procedure IntializeRule(examples){
  return 'true → '
}

procedure SelectCandidates(rules, examples){
  return rules // Wähle alle Regeln aus
}

procedure FilterRules(rules, examples){
  return rules // AQ Filtert keine Regeln
}
```

Quelltext 4.2 Nicht-triviale Funktionen des AQ-Algorithmus

```
procedure RuleStoppingCriterion(theory, rule, examples){
  return AqrRuleStop(theory, rule, examples) // Noch zu Def.
}

procedure EvaluateRule(rule){
  return AqrDifference(rule) // Noch zu Definieren
}

procedure RefineRule(candidates, examples){
  refinements := ∅
  find seed
  find E_neg from examples
  EXTENSIONS := createExtensions(seed, E_neg)
  foreach extension ∈ EXTENSIONS
  {
    newrule := candidate ∪  extension
    refinements := refinements ∪ newrule
  }
  return refinements
}

procedure StoppingCriterion(theory, rule, examples){
  return coveredNegatives == ∅
}
```

4.3.1 Die Implementierung des AQ

Wie wird aber nun der AQ-Algorithmus wirklich umgesetzt? Vorhanden ist das wissen, welche Funktionen der SeCo-Factory wie durch den AQ-Algorithmus erledigt werden müssen. Wie das geschieht, wird jetzt anhand von einigen Beispielen gezeigt. Alle in den Implementierungen vorkommenden Variablennamen die mit $m\_$ beginnen, sind Variablen, die über Parameter in der XML-Datei geändert werden können. Schon in Quelltext 4.2 konnte man sehen, dass wir einige angepasste Heuristiken benötigen. So wird z.B. AqrDifference, AqrStopping und AqrRuleStop benötigt. Dabei wird AqrRuleStop die Implementierung für das RuleStoppingCriterion, AqrStopping die Implementierung für das StoppingCriterion und AqrDifference eine Heuristik, die benötigt wird in EvaluteRule.
Der Unterschied des AqrDifference zu einem normalen Difference (siehe [2]) ist, dass im Falle von weniger als einem abgedeckten Positiv-Beispiel als Bewertungs-

Quelltext 4.3 evaluateRule Funktion des AqrDifference

```
public double evaluateRule(CandidateRule r) {
  TwoClassStats t = r.getStats();
  double P = t.getIsPositive();
  double N = t.getIsNegative();
  double p = t.getTruePositive();
  double n = t.getFalsePositive();

  if(p < 1.0) {
    return (-N);
  }
  // normalcase
  return (p - n);
}
```

wert $(-N)$ zurückgegeben wird, wobei N für die Anzahl an Negativ-Beispiele innerhalb der Trainingsmenge steht. Mit der Bewertung $(-N)$ erreicht man, dass Regeln, die weniger als ein positives Beispiele abdecken, den schlechtesten erreichbaren Wert bekommen, so dass alle anderen Regeln immer besser sein werden, als eine Regel, die keine neuen positiven Beispiele abdeckt.
Die evaluateRule Funktion des AqrDifference wird in Quelltext 4.3 dargestellt.
Das AqrStopping-Criterion stoppt das Verfeinern einer Regel, wenn keine negativen Beispiele mehr abgedeckt werden. Die genaue Anzahl kann als Parameter threshold geändert werden. Dieser Parameter heisst im Code *m_threshold*, da alle Variablen, die über Parameter geändert werden können ein *m_* im Namen vorgestellt bekommen. Die checkForStop Funktion wird in Quelltext 4.4 dargestellt.
Das AqrRuleStop unterscheidet sich nur in seiner Nutzung der Heuristik. Da im AqrRuleStop als Heuristik die AqrDifference benutzt wird, muss sie selbst geschrieben werden. Der Teil, der sich von DefaultRuleStop unterscheidet, ist in Quelltext 4.5 angegeben.
Bei einer anderen Funktion des SeCo-Frameworks dauert die Umsetzung des AQR-PseudoCodes in realen Code länger. Jetzt fehlt noch der RuleRefiner. Der RuleRefiner übernimmt die Verfeinerung der Regeln und gibt alle verfeinerten Regeln zurück. Einige weitere Aufgaben neben der reinen Verfeinerung von Regeln muss der RuleRefiner des AQR-Algorithmus erfüllen. Zunächst muss ein sogenanntes **Seed** für eine Regel bestimmt werden. Dies geschieht nur einmalig pro Generierung von neuen Regeln. Sollen nur die vorhanden Regeln verbessert werden, wird das alte Seed weiter verwendet. Dann muss ein Negativ-Beispiel mit dem geringsten Abstand (der geringsten Entfernung) gefunden werden; denn je kleiner der Abstand, desto weniger Attribute unterscheiden das Seed und das

Quelltext 4.4 checkForStop Funktion des AqrStopping-Criterion

```
public boolean checkForStop(seco.models.CandidateRule refinement,
    weka.core.Instances examples) throws Exception {
  TwoClassStats stats1 = refinement.getStats();
  double p = stats1.getTruePositive();
  double n = stats1.getFalsePositive();

  boolean result = false;
  if (n <= (int) m_threshold){
    result = true;
  }
  // if no more negativ examples are covered,
  // dont refine this rule!
  return result;
}
```

Quelltext 4.5 AqrRuleStop

```
public AqrRuleStop(){
  ...
  m_ruleEvaluator = new DefaultRuleEvaluator();
  m_ruleEvaluator.setHeuristic(new seco.heuristics.AqrDifference());
  ...
}
```

Negativ-Beispiel. Aus dem Seed und dem Negativ-Beispiel müssen die sogenannte *Extensions* generiert werden. Es wird in RefineRule außerdem ein *Star* generiert, der alle Verfeinerungen enthält und in dem alle verfeinerten Regeln solange verfeinert werden, bis sie das Negativ-Beispiel nicht mehr abdecken. Außerdem werden nur *maxStar* viele neue Regeln generiert. Der Paramter *m_ negativeSelection* ist über die XML-Konfigurationsdatei beeinflussbar. RefineRule des AqrRefinerTop-Down sieht dann wie in Quelltext 4.6 dargestellt aus:

Quelltext 4.6 RefineRule des AqrRefinerTopDown

```
public RuleSet refineRule(CandidateRule c, Instances examples)
 throws Exception {
  // choose the new Seed, or return the old seed
  Instance aqrSeed = createSeed(examples, m_seedChoice, c);

  // find all covered Negative examples,
  // delete all covered positive examples
  Instances coveredNegatives =
   c.coveredNegInstances(examples, m_classVal);

  RuleSet star = new RuleSet();

  // behavior of the Selection of the negative Examples
  // 0 = distance, 1 = random
  if(m_negativeSelection == 0){
    star = createStarByDistance(c, aqrSeed, coveredNegatives);
  } else {
    star = createStarByRandom(c, aqrSeed, coveredNegatives);
  }
  return star;
}
```

Mit *m_ negativeSelection* kann man auswählen wie das Negativ-Beispiel gewählt wird; Entweder (wie im Original Algorithmus vorgesehen) über den Abstand oder ob es zufällig gewählt wird. Welche der beiden Methoden bessere Ergebnisse liefert, wird später noch getestet. Ein Diagramm zur Verdeutlichung der Aufgaben der RefineRule-Funktion des AQ-Algorithmus findet sich in Abbildung 4.2. Auf die genaue Implementierung von createSeed, createStarByDistance und createStarByRandom wird nicht näher eingegangen.

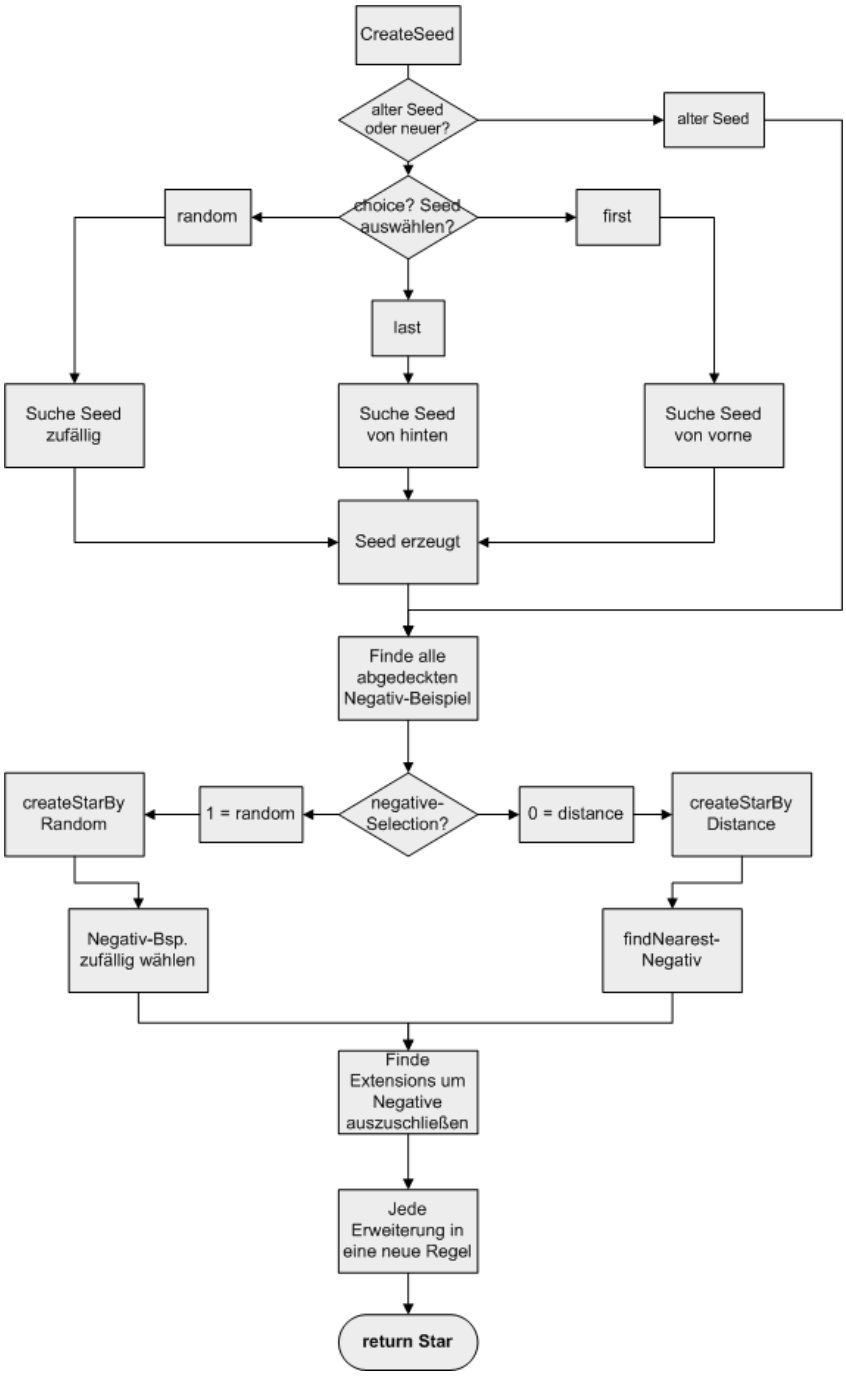


Abbildung 4.2: Aktivitätsdiagramm von RefineRule des AQ

createSeed Diese Methode versucht, je nach gewünschter Vorgehensweise, ein Positiv-Beispiel zu finden, das als Seed genutzt wird. Dafür wertet es die Variable choice aus, die drei Werte annehmen kann: „first, last und random“. Bei choice=first sucht die Methode von vorne nach hinten (durch die Beispiele) nach

einem Beispiel, das der gesuchten Klasse entspricht. Wenn die Methode ein Beispiel findet, dann wird dieses als Seed zurückgegeben. Bei choice=last wird die Beispielmenge von hinten nach vorne nach einem Seed durchsucht. Als letzte Variante gibt es choice=random, welches auch das Default-Suchverhalten ist. Dort wählt die Methode zufällig ein Beispiel aus und überprüft, ob das Beispiel der gesuchten Klasse entspricht. Wenn ja wird dieses Beispiel als neuer Seed zurückgeliefert. Diese Methode überprüft auch, ob ein neues Seed notwendig ist oder ob das alte Seed wieder genutzt werden muss. Ein neues Seed ist immer notwendig, wenn neue Regeln generiert werden. Das alte Seed wird für die Verfeinerung der alten Regeln benötigt.

createStarByDistance Wie bei *createStar* ist das Ziel dieser Methode einen *Star* mit Regeln zu generieren. Der Unterschied zu createStarByRandom besteht in der Form, wie diese Funktion die negativen Beispiele auswählt. Während bei createStarByRandom das Negativ-Beispiel in keinerlei Beziehung zu dem Seed steht, ist es nun bei createStarByDistance so, dass das Negativ-Beispiel abhängig von der Entfernung zum Seed gewählt wird. Dies geschieht über die oben vorgestellte Methode findNearestNegative. Ansonsten geht es genauso weiter wie bei createStarByRandom: Die Funktion createExtensions wird aufgerufen, welche sich um die Extensions für die Regeln kümmert. Danach werden aus jeder Regel n neue Regeln (bei n Extension-Attributen) erstellt. Denn jedes Attribut einer Extension wird mit einer Regel zusammenführt und ergibt dann eine neue Regel. Bei x Regeln und y Extension-Attributen ergibt das also $x \times y$ neue Regeln, die in den Star aufgenommen werden müssen. Diese Regelmenge wird dann zurückgegeben.

createStarByRandom Ziel dieser Methode ist, einen *Star* zu generieren. Dieser besteht aus Regeln, die alle das Seed abdecken aber kein Negativ-Beispiel. Dies geschieht, indem die Hauptschleife solange durchlaufen wird, bis kein Negativ-Beispiel mehr abgedeckt wird. In der Hauptschleife wird dann zufällig[4] ein Negativ-Beispiel aus der Menge aller Negativ-Beispiele gezogen. Nun wird die Funktion createExtensions aufgerufen, welche sich um die Extensions für die Regeln kümmert. Dann werden aus jeder Regel n neue Regeln (bei n Extension-Attributen) entstehen. Denn jedes Attribut einer Extension wird mit einer Regel zusammengeführt und ergibt dann eine neue Regel. Bei x Regeln und y Extension-Attributen entstehen also $x \times y$ neue Regeln, die in den Star aufgenommen werden. Diese Regelmenge wird dann zurückgegeben. Generell werden nur die Regeln verfeinert, die noch negative Beispiele abdecken.

findNearestNegative Die Methode hat als Ziel, ein Negativ-Beispiel zu finden, das sich möglichst wenig von dem Seed unterscheidet. Dies wird erreicht, indem die Attribute des Seeds und des Negativ-Beispiels verglichen werden. Je

[4]Unterschied zu createStarByDistance

mehr Attribute sich unterscheiden, desto größer ist die **distance**, also der „Abstand“ zwischen den Beiden. Ideal ist ein Abstand von eins, d.h. dass sich nur ein Attribut unterscheidet und man mit Hilfe dieses einen Attributes zwischen der Klasse Positiv und der Klasse Negativ unterscheiden kann. Ein Abstand von 0 (also kein Attribut unterscheidet sich) ist denkbar schlecht, da es somit nicht möglich ist ein Attribut anzugeben, welches eine Unterscheidung erlaubt. Normalerweise wird dann das Negativ-Beispiel ignoriert und das nächste negative Beispiel wird ausgewählt. In findNearestNegative wird versucht ein Negativ-Beispiel mit Abstand >0 zu finden. Falls das nicht geht, wird das Beispiel mit Abstand 0 zurück gegeben.

createExtension Diese Methode erhält den Seed und das ausgewählte Negativ-Beispiel als Eingabe. Die Aufgabe dieser Methode ist nun alle möglichen Attribute zu finden, die eine Unterscheidung zwischen dem Seed und dem Negativ-Beispiel zu ermöglichen. Falls keine Attribute zur Unterscheidung existieren, wird nichts zurückgegeben. Tabelle 4.6 zeigt einige Beispiele zur Findung der Extensions.

Tabelle 4.6: Extension Beispiele

Beispiel	Outlook	temperature	humidity	windy	class
Seed	sunny	hot	high	false	no
Negativ	overcast	hot	high	false	yes
Extension	sunny	-	-	-	-
Seed	sunny	hot	high	false	no
Negativ	sunny	cool	normal	true	yes
Extension	-	hot	high	normal	false

4.3.2 AQ XML-Datei

Wie solch eine XML-Datei (für den AQR-Algorithmus) aussehen könnte, sieht man in Tabelle 4.7.

Tabelle 4.7: SeCo XML-Konfigurationsdatei

```
<seco>
  <secomp interface="ruleevaluator" classname=
   "DefaultRuleEvaluator">
    <jobject package="seco.heuristics" classname=
     "AqrDifference" setter="heuristic"/>
  </secomp>

  <secomp interface="rulerefiner" classname=
   "AqrRefinerTopDown" package="seco.learners.aqr">
    <property name="negativeSelection" value="0"/>
    <property name="seedChoice" value="random"/>
  </secomp>

  <secomp interface="selector" classname="DefaultSelector"/>

  <secomp interface="stopcriterion" classname="AqrStopping">
    <property name="threshold" value="0.0"/>
  </secomp>

  <secomp interface="rulestopcriterion" classname="AqrRuleStop"/>

  <secomp interface="rulefilter" classname="MultiRuleFilter">
    <jobject classname="BeamWidthFilter" setter="filter">
      <property name="beamwidth" value="3"/>
    </jobject>
  </secomp>

  <property name="skipdefaultclass" value="true"/>
</seco>
```

Wie man in der XML-Datei für den AQR-Algorithmus sehen kann, wird als RuleEvaluator die Default-Variante genommen, jedoch wird als Heuristik die AqrDifference statt Laplace genommen.

Erkennbar wird hier die Übergabe der Parameter:

- für die maximale Größe eines Star (*Beamwidth*),
- für die Entscheidung, wie die negativen Beispiele ausgewählt werden sollen (*negativeSelection*), und
- wie das Seed-Beispiel ausgewählt werden soll (*seedChoice*).

Welche Parameter man genau an jede Komponente übergeben kann, steht definitiv im Programmcode und auch im JavaDoc der Seco-Api. Gesetzt den Fall man will probieren, wie sich der Algorithmus mit einem anderen stopcriterion verhält, muss man in der XML-Konfigurationsdatei nur das alte (AqrStopping) durch dieses neue stopcriterion ersetzen. Das neue stopcriterion muss gewisse Kriterien erfüllen, um Endlosschleifen zu vermeiden. Welche das sind und warum das so ist, wird nachfolgend erklärt. Das neue Stopping Criterion muss den möglicherweise auftretenden Fall auffangen, dass eine Regel gelernt wird, die ein negatives und ein positives Beispiel abdeckt. Das negative kann jedoch nicht entfernt werden. Diese Gefahr besteht für alle „Rest-Mengen“ die sich nicht mehr trennen lassen. Da die SeCo-Factory keine Möglichkeit vor sieht in solchen Fällen einfach abzubrechen, wurde diese Spezialfall-Behandlung als Anker eingebaut. Der Anker dient zur Minimierung der Gefahr von Endlos Refiningschleifen. Solche Fälle treten (wenn überhaupt) nur gegen Ende des Lernprozesses auf, wo z.B. nur noch dieses eine positive Beispiel übrig geblieben ist. Dieser Fall ist sehr selten, kann aber auftreten. Deswegen wurden alle ähnlich gelagerten Fälle, wie ein positives und zwei negative, oder zwei positive und ein negatives etc., nicht explizit betrachtet Hier wäre eine Änderung der SeCo empfehlenswert.

5 Weighted Covering

In diesem Kapitel wird erläutert was Weighted Covering ist, was für Vorteile es möglicherweise bringen kann und wie die SeCo-Factory erweitert wurde, um das Weighted Covering zu unterstützen. Außerdem gehen wird darauf eingegangen, wie eigene Algorithmen vorbereitet werden müssen um „Weighted Covering“ zu unterstützen. Zum Schluß wird gezeigt, wie Weighted Covering in der SeCo-Factory zu nutzen ist, insbesondere wird auf die Konfiguration der XML-Datei eingegangen.

5.1 Was ist Weighted Covering?

Die Idee hinter Weighted Covering ist, dass man versucht, Beispiele mit Gewichten zu versehen und so dem Lern-Algorithmus mehr Möglichkeiten bietet, gute Regeln zu lernen. Durch die Gewichtung könnten z.B. Meta-Informationen dem Lerner indirekt zur Verfügung gestellt werden indem Beispiele mehrmals vorkommen und somit ein höheres Gewicht für den Lern-Algorithmus haben. Eine andere Möglichkeit ist die Beispiele dem Lernalgorithmus zu erhalten und nicht abgedeckte Beispiele aus der Trainingsmenge nicht zu entfernen, sondern deren Gewichtung zu ändern. Ein Vorteil des gewichteten Lernens ist, dass man es auch einsetzen kann, wenn man den Lern-Algorithmus nicht ändern kann; denn wichtige Beispiele können einfach mehrfach der Trainingsmenge hinzugefügt werden, was einer höheren Gewichtung dieser Beispiele entspricht. In dieser Arbeit wird lediglich auf die Möglichkeit jedem Beispiel e_i das Gewicht w_i zu geben eingegangen.
Was bewirkt dieses Gewicht? Man betrachte eine Menge an Trainingsbeispielen, da sich ein Algorithmus am Anfang des Lernens befindet, sind alle Beispiele mit einem Default-Wert (meist 1.0) ausgestattet. In Abbildung 5.1 sieht man ein Beispiel wie das Gewichten Einfluss auf die Regelfindung haben kann. Regel R2 hätte wahrscheinlich mit Covering nicht gefunden werden können. Vielleicht will man aber eine hohe Abdeckung an Positiv-Beispielen haben, was mit normalem Seperate and Conquer nicht zu erreichen ist.
Man betrachte sich ein Beispiel mit 10 Trainingsbeispiele und 2 Klassen. Fünf sind positiv und fünf negativ. Am Anfang haben alle Beispieldaten das Gewicht 1.0. Die erste Regel würde drei Positiv-Beispiele und ein Negativ-Beispiel abdecken. Dann werden die Gewichte der drei positiv- und des Negativ-Beispiels verringert. Dem Lernalgorithmus stehen jetzt zum Lernen immer noch alle Beispiele zur Verfügung, auch wenn vier von den 10 Trainingsbeispiele ein anderes Gewicht haben.

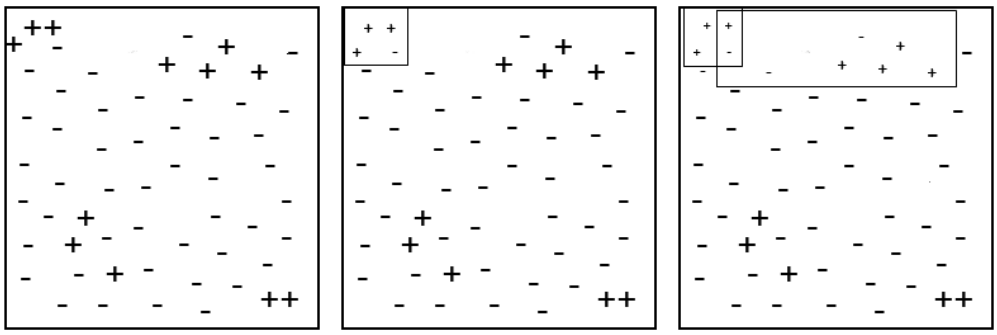


Abbildung 5.1: Weighted Covering Beispiel

5.2 Möglichkeiten der Gewichtung

Welche Möglichkeiten gibt es Beispiele zu gewichten?
Es gibt zwei Richtungen:

- 1. Das Gewicht der **abgedeckten** Beispiele verändern
- 2. Das Gewicht der **nicht abgedeckten** Beispiele verändern

Bei Möglichkeit 1 wird meistens das Gewicht der abgedeckten Beispiele verringert, um diesem bei erneutem Abdecken weniger Einfluss zu geben. Dieses Verringern kann z.B. ein einfaches Halbieren des Gewichtes sein. Ein Gewichtsverlauf würde bei zweimaligem Abdecken wie der Folgende aussehen: $1.0 \rightarrow \frac{1}{2} \rightarrow \frac{1}{4}$. Oder es besteht die Möglichkeit (wie in [7] vorgestellt), das Gewicht der abgedeckten, aber falsch klassifizierten, Beispiele zu erhöhen, so dass diese bei erneuter Abdeckung als Negative mehr ins Gewicht fallen und die Regel somit schlechter bewertet wird.
Bei Möglichkeit 2 wird meist das Gewicht aller Beispiele, die nicht durch die Regel abgedeckt wurden, erhöht. Zur Erhöhung kann das Gewicht durch Multiplikation verdoppelt oder durch Addition eines Terms verdoppelt werden. Die Variante der Gewichtung der nicht abgedeckten Beispiele wird in dieser Arbeit nicht behandelt.

Einigen konkrete Vorschläge zur Gewichtung von Beispielen aus der Literatur: SLIPPER, Lightweight Rule Induction und die Vorschläge von Lavrač et al.

5.2.1 SLIPPER

In [5] wird der SLIPPER-Algorithmus vorgestellt. Dieser Algorithmus arbeitet mit Boosting und ist für Zwei-Klassen-Probleme implementiert. Jede gelernte Regel dieses Algorithmus hat einen so genannten Confidence Faktor, (Zuversichts-Faktor). Diese Zahl gibt an wie „sicher“ die Regel bei der Klassifizierung ist. SLIPPER baut auf zwei Lerner auf: eine sog. schwache Hypothese (ein Regellerner) und eine starke Hypothese. Der Regellerner lernt eine Regel, die es erlaubt

die Trainingsdaten in zwei Teile aufzusplitten. Die Daten, die die Regel erfüllen und die, die es nicht tun. Beim Klassifizieren ergibt diese Regel einen Wert von 0, falls die Regel ein neues Beispiel abdeckt, und einen Wert größer 0, falls die Regel zutrifft. Dieser Confidence Wert C_R ist für alle Beispiele $x \in R$, die von der Regel abgedeckt werden, gleich groß. Also gilt für jede Regel t: $\forall x \in R_t, a_t h_t(x) = C_{R_t}$, wobei $h_t(x)$ die Vorhersage der Regel t ist und a_t ein Faktor für die Gewichtung. Um ein Beispiel x mit der starken Hypothese zu klassifizieren, addiert man einfach alle C_{R_t}, die x erfüllen, und trifft die Vorhersage basierend auf dem Vorzeichen der Summe. Der Gewichtungsfaktor a_t wird nach jedem Regellernschritt basierend auf der Verteilung der Beispiele in Abhängigkeit ihrer bisherigen Klassifizierung geändert. Die abgedeckten Beispiele bekommen ein geringeres Gewicht, wobei das Ausmaß der Gewichtsreduktion von der Genauigkeit der neuen Regel abhängt.

5.2.2 Lightweight Rule Induction

In [7] wird der Lightweight Rule Induction Algorithmus vorgestellt, der kompakte DNF-Regeln generiert. Diesen Algorithmus zeichnet aus, dass für jede Klasse die gleiche Anzahl an ungewichteten Regeln gelernt werden. Die Klassifikation neuer Beispiele erfolgt durch eine Art Wahl. Jede Regel stimmt ab, welcher Klasse das neue Beispiel entsprechen könnte. Die Ergebnisse werden gezählt und die Klasse, die am Ende die meisten „Stimmen“ hat, gewinnt. Somit wird das neue Beispiel dann die Klasse des „Siegers“ der Wahl bekommen. Das geschieht alles demokratisch, d.h. jede Regel hat nur eine Stimme und jede Klasse hat die gleiche Anzahl an Regeln. Dieser Algorithmus gewichtet, indem das Gewicht der falsch klassifizierten Beispiele erhöht wird. Hierbei sei $e(i)$ die Anzahl der Regeln, die das Beispiel i bisher falsch klassifiziert haben. Das Gewicht für ein Beispiel i berechnet sich wie folgt:

$$weight(i) = 1 + e(i)^3$$

Dieser Ansatz wird später in Kapitel 5.2.7 aufgegriffen.

5.2.3 Vorschläge von Lavrač

In [8] geht Lavrač auf den Ansatz des so genannten *Subgroup Discovery* ein. Subgroup Discovery und das Standard-Regel-Lernen unterscheiden sich in der Art und Weise, wie Regeln gesucht werden. Normales Regel-Lernen zielt darauf ab, für jede Klasse ein Modell zu lernen. Dabei werden dann die Klassencharakteristiken anhand von Eigenschaften der Trainingsbeispielen festgestellt. Subgroup Discovery im Gegensatz zielt darauf ab, einzelne Regeln oder Muster, die von Interesse sein könnten, zu finden. Nada Lavrač hat in ihrer Arbeit den CN2 Algorithmus um diese Möglichkeit erweitert. In ihrem Ansatz gibt sie jedem positivem Beispiel ein Gewicht von 1. Also für jedes positive Beispiel e_j wird das Gewicht gesetzt mit $weight(e_j, 0) = 1$, wobei 0 dafür steht, dass dieses Beispiel 0

mal abgedeckt wurde. Der Gewichtungswert von 1 entspricht somit der Aussage „Bitte decke mich ab“, wobei ein Wert $0 < weight < 1$ für „Strenge dich nicht an, mich abzudecken“ steht. Dabei ist die Anstrengung fließend Anti-proportional zu dem Gewichtungswert, also je näher an der 1, desto weniger Aufwand wird betrieben. Die Gewichtung eines Beispiels kann somit nur Werte zwischen 0 und 1 annehmen. Lavrač hat dabei zwei verschiedene Gewichtungsmethoden vorgeschlagen.

5.2.4 Multiplikative Gewichte

Es wird ein Parameter γ mit einem Wertebereich von $0 < \gamma < 1$ eingeführt. Die Gewichte von abgedeckten positiven Beispielen e_j verringern sich dann wie folgt:

$$weight(e_j, i) = \gamma^i$$

Hierbei ist i die Anzahl an Abdeckungen für dieses Beispiel. Mit einem Wert für γ gleich 0, was außerhalb des Wertebereichs liegt, würde das Verhalten dem eines „normalem“ CN2 Algorithmus entsprechen, da der Algorithmus Beispiele mit dem Gewicht 0 ignoriert. Könnte γ den Wert 1 annehmen, würde immer wieder die selbe Regel gefunden werden, da sich das Gewicht der abgedeckten Beispiele nicht ändern würde.

5.2.5 Additive Gewichte

Bei dieser Methode verringern sich die Gewichte der abgedeckten positiven Beispiele nach folgender Formel:

$$weight(e_j, i) = \frac{1}{1+i}$$

Wobei i für die Anzahl der Abdeckungen durch Regeln steht.
In der ersten Iteration haben alle positiven Beispiele das gleiche Gewicht:

$$weight(e_j, 0) = \frac{1}{1+0} = 1$$

Hier ist $i = 0$, denn in der ersten Iteration deckt noch keine Regel ein Beispiel ab.

5.2.6 Multiplikatives Gewichtungsmodell

Dieses einfach Modell verringert nur das Gewicht aller abgedeckten Beispiele um die Hälfte, also

$$weight(e_j, i) = w(e_j, i-1) \cdot \frac{1}{2}$$

Eigentlich ist der Faktor, um den das alte Gewicht verringert wird, wieder als Paramter wählbar, zur einfacheren Übersicht wurde er hier aber als $1/2$ gewählt. Ein Ausschnitt aus diesem Modell gibt es in Quelltext 5.6 zu sehen.

5.2.7 Gewichtungsmodell von LRI

Die Formel, die dem Modell *WeightNegativeToThePower* zugrunde liegt, wurde in [7] erörtert. Der Grundgedanke ist Folgender: Wenn ein oder mehrere Regeln ein Beispiel falsch klassifizieren, sollte diese falsche Klassifikation die Bedeutung dieses Beispiels erhöhen. Diese Vorgehensweise wurde vom *Boosting* übernommen. Weiss und Indurkhya kamen in [7] zu dem Schluss, dass eine Potenzierung mit einem festen Faktor eine extreme Auswirkung auf die abgedeckten Beispiele haben sollte. Sie zählten die Anzahl der Regeln, die ein Beispiel i falsch klassifizierten. Dann berechneten sie das Gewicht, wie in Kapitel 5.2.3 angegeben, nach der Formel:

$$weight = 1 + e(i)^3$$

$e(i)$ stellt die Anzahl der Regeln dar, die das Beispiel **i** falsch klassifiziert haben. Ein Beispiel zur Lightweight-Gewichtungs-Methode findet sich in Tabelle 5.1 .

Tabelle 5.1: Beispiel zur Lightweight Gewichtung

10 Regeln bisher gelernt, davon klassifizieren 4, inklusive der aktuellen Regel, das aktuelle Beispiel i falsch. Das ergibt ein Gewicht für dieses Beispiel von

$$weight = 1 + 4^3$$

$$weight = 65$$

Also bekäme das aktuelle Beispiel ein Gewicht von 65, was bedeutet, dass dieses Beispiel den gleichen Wert hat, wie 65 Beispiele mit normalem Gewicht. Eine Regel, die dieses Beispiel also nochmal falsch klassifiziert, müsste sehr viele richtig klassifizierte Beispiele finden um überhaupt gegen andere Regeln, die dieses Beispiel nicht abdecken, eine Chance zu haben.

5.2.8 Vorteile von Weighted Covering

Aber was sind die Vorteile des Weighted Covering gegenüber dem üblichen Ansatz des Entfernens von abgedeckten Beispielen? Bei dem normalem Entfernen der abgedeckten Beispiele fehlen diese dem Lernalgorithmus. Da normalerweise aus einem Refining (eine Verbesserung einer Regel) immer nur eine „beste Regel“ benutzt wird, besteht die Möglichkeit, dass andere Regeln, die nur etwas schlechter als die „beste Regel“ waren, aber noch andere Beispiele abdeckten, nicht mehr gefunden werden. Da die Beispiele, die zu dieser Regel geführt hatten, nicht mehr vorhanden sind. Dies ist vor allem dann von Nachteil, wenn die ersten Regeln sehr kurz, also mit wenigen Attribut-Tests, waren. Der Algorithmus hat zwar immer

noch die Möglichkeit weitere kurze Regeln zu finden, die auch schon abgedeckte Beispiele abdecken, aber diese schon abgedeckten Beispiele werden für diese Regel nicht gewertet, somit wird diese Regel von den Bewertungsfunktionenen meist schlechter bewertet. Vor allem entgeht dem Algorithmus somit die Möglichkeit Redundanz in die Theorie einzubringen, was dann von Vorteil ist, wenn die Daten nicht vollkommen korrekt sind. Der Einfluss einer einzelnen falsch gelernten Regel sinkt damit. Die Mehrfachabdeckung steigert auch die Fehlertoleranz. Ob ein Algorithmus durch Weighted Covering genauer wird und ob der Algorithmus kürzere Regeln findet, wird sich in der Evaluierung in Kapitel 6 zeigen. Leider wird aber das Weighted Covering, das volle Potenzial noch nicht ausspielen können, da die Bewertungsfunktionen der SeCo nur nach der ersten feuernden Regel klassifizieren. Sie müssten dahingehend geändert werden, dass die Redundanz (die man sich von Weighted Covering verspricht) auch zum Tragen kommen kann. Empfehlenswert wäre eine Klassifizierung nach der Menge der klassifizierenden Regeln. Also das die Klasse „gewinnt“ für die mehr Regeln sprechen.

5.3 Erweiterung der SeCo-Factory um Weighted Covering

Als erstes wird darauf eingegangen, wie die SeCo-Factory erweitert wurde, um Weighted Covering zu unterstützen. Die Original-Factory ist nicht ideal für das Weighted Covering geeignet, sie musste angepasst werden. Zuerst wurde zur SeCo-Factory ein weiteres Interface hinzugefügt, das für die Gewichtung der Beispiele zuständig sein sollte. Dieses Interface IWeightModell dient dem Benutzer zur Bereitstellung eines eigenes Gewichtungsmodells. In Quelltext 5.1 und 5.2 ist die Interface-Definition von changeWeights zu sehen.

Quelltext 5.1 Interface des IWeightModell Teil1

```
/**
 * The changeWeights change the weight
 * of the examples covered by rule.
 * @param examples
 *   The Examples where the weight have to be changed
 * @param covered
 *   This is a container for counting how many rules cover a example
 * @param rule
 *   The rule, that initiate the "covering" and so the new weights
 * @return
 *   A array of the weighted examples (0) and the cover counter (1)
 * @throws Exception
 */
public Instances[] changeWeights(Instances examples, Instances covered,
  CandidateRule rule)throws Exception;
```

Quelltext 5.2 Interface des IWeightModell Teil2

```
/**
 * The calcNewWeight method calculates the weight of
 * the delivered example
 * @param example
 *   The example where the weight have to be updated
 * @param cover
 *   The container-example with the count of covered rules
 * @return
 *   The weight of the Example
 * @throws Exception
 */
public double calcNewWeight(Instance example, Instance cover)
  throws Exception;
```

Als nächstes musste dieses Interface der SeCo-Factory auch bekannt gemacht werden. Dazu wurde im Package seco.learners.factory die Klasse ConfigurableSeco um einige Einträge erweitert. Dazu gehört z.B. der Vergleichsstring für die XML Interface Erkennung:
`public final static String WEIGHTCRITERION = "weightcriterion";`.
Dieser wird benötigt um das Interface des Weighted Covering in der XML Datei

angeben zu können. In Quelltext 5.3 ist zu sehen wie die Methode setSecoComponent an dieser Stelle geändert wurde.

Quelltext 5.3 Änderung an setSeCoComponent

```
public void setSecoComponent(ISecoComponent secomp, String ifaceName)
{
  if ( ifaceName.equals( SELECTOR ))
  ...
  else if ( ifaceName.equals( STOPCRITERION ) )
  {
    m_stopCriterion = (IStoppingCriterion)secomp;
  }
  // following added by M. Ruppert
  else if (ifaceName.equals(WEIGHTCRITERION)){
    m_weighter =(IWeightModell)secomp;
  }
}
```

Hier wurde der SeCo-Factory die Möglichkeit gegeben, die Übergabe der Implementierung aus der XML Datei zu verarbeiten.

Als nächstes musste die abstrakte Klasse AbstractSeco des packages seco.learners.core angepasst werden. Hier wurde einerseits ein Boolean Parameter eingeführt, der es erlaubt, Weighted Covering in der XML Datei zu aktivieren. Die AbstractSeco wurden dahingehend geändert, dass bei gewünschtem Weighted Covering die Gewichtungsmethode changeweights aufgerufen wird. Andererseits wurde eine Möglichkeit vorgesehen, für jedes Beispiel festzuhalten, wie oft es von einer Regel abgedeckt worden ist. Dies wird für einige Gewichtungsansätze interessant sein. Sehr wichtig ist auch das Erweitern der Methode setClassVal um den in Quelltext 5.4 gezeigten Code, denn sonst steht der Gewichtungsfunktion die Klasse, die gelernt werden soll, nicht zur Verfügung.

Quelltext 5.4 Erweiterung der Methode setClassVal

```
...
  compName = "IStopCriterion";
  setupComponentWithClassVal(m_stopCriterion, m_classVal);
  compName = "IWeightModell"; // neu hinzugefügt
  setupComponentWithClassVal(m_weighter, m_classVal);
  ...
```

Es wurde außerdem noch die Klasse Rule des packages seco.models um eine Funktion erweitert, die es ermöglicht, sich nur die abgedeckten Negativ-Beispiele zu-

rückgeben zu lassen. Diese Methode heißt coveredNegInstances.

5.4 Weighted Covering im eigenen Algorithmus

Ein Design-Ziel der Implementierung des Weighted Covering in der SeCo-Factory war, dass man an einem bestehenden Algorithmus möglichst wenig ändern muss um Weighted Covering verwenden zu können. Es muss aber beachtet werden, dass die Heuristiken und Kriterien darauf ausgelegt sein müssen. So kann nicht einfach das Standard rulestopcriterion eines Algorithmuses verwendet werden., denn beim Weighted Covering übernimmt das rulestopcriterion auch die Aufgabe zu entscheiden, ob der Lernalgorithmus fertig ist. Im Unterschied zum Standard Separate and Conquer Algorithmus, wo spätestens dann aufgehört wird, wenn es für die aktuelle Klasse keine Positiv-Beispiele mehr in der Trainingsmenge gibt, greift dieses Austiegskriterium beim Weighted Covering nicht, da ja keine Beispiele entfernt werden! Deswegen wird hier in Quelltext 5.5 ein Beispiel für das gewichtete Stop-Kriterium des AQR-Algorithmus gezeigt.

Quelltext 5.5 Ausschnitt aus AqrRuleStopWeighted

```
public boolean checkForRuleStop(RuleSet theory, CandidateRule rule,
  Instances examples, Instances covered) throws Exception
{
  CandidateRule defaultRule;
  defaultRule = new CandidateRule( new NominalCondition(
   examples.classAttribute(), m_classVal ));
  m_ruleEvaluator.evaluateRule(defaultRule, examples );
  m_ruleEvaluator.evaluateRule(rule, examples );
  if ( rule.length() == 0 ) return true; // rule stop, if there
    // are no conditions
  boolean result =
    CandidateRule.betterRule(rule, defaultRule) != rule;
  Instance testForCount;
  for(int i= 0; i < covered.numInstances(); i++){
    testForCount = covered.instance(i);
    if (testForCount.classValue() == m_classVal){ // go through
      //all positves
      // if one example is covered fewer times than
      // "positiveHasToCover" don't stop
      if(covered.instance(i).weight() < m_positiveHasToCover){
        return false;
      } else {
        result = true;
      }
    }
  }
  return result;
}
```

Die Variable *m_positiveHasToCover* wird als Parameter in der XML-Konfiguration angegeben. Der Default-Wert für *m_postiveHasToCover* ist 1.0, was soviel bedeutet wie das jedes Positiv-Beispiel mindestens einmal abgedeckt werden muss.Deswegen ist es wichtig, dass in jedem Gewichtungsmodell die Anzahl der abdeckten Regeln eines Beispiels angepasst werden. Es wurde in Abschnitt 5.2.6 ein einfaches Modell vorgestellt, dass nun implementiert wird.

5.4.1 Implementierung des einfachen Gewichtungsmodells

Hier wird der Pseudocode des in 5.2.6 erwähnten Modells vorgestellt. Als erstes wird der Pseudocode der Implementierung vorgestellt, anschließend die Erklärung des Pseudocodes.

Quelltext 5.6 Einfaches Gewichtungsmodell WeightPerIteration

```
procedure changeWeights(examples, rule){
  foreach coverTest ∈  examples{
    if(rule.covers(coverTest)){
      newweight = calcNewWeight(coverTest)
      coverTest.weight = newweight
      coverTest.covered++
    }
  }
  return examples
}

procedure calcNewWeight(example){
  weight = example.weight · MultiValue
  weight = weight + AddValue
  return weight
}
```

In diesem Gewichtungsmodell ist es vorgesehen zwei Parameter in der XML Datei anzugeben: *MultiValue* und *AddValue*. Das *MultiValue* ist als Faktor für die Multiplikation der Gewichte gedacht. Äquivalent dazu wirkt sich das *AddValue* als Additionsfaktor für die Gewichtung aus. Zur Kalkulation des neuen Gewichtes wurde die Methode calcNewWeight eingeführt um die Berechnung des Gewichtes von der Zuweisung zu trennen. Um das Gewicht eines Beispiels zu halbieren, wird das *Multivalue* auf *0.5* und das *AddValue* auf *0.0* gesetzt.

5.4.2 Erläuterung von WeightPerIteration

Aber wie funktioniert nun die Gewichtung mit dem Gewichtsmodell nach Pseudocode 5.6? Als erstes die Betrachtung der äußeren Schleife:

```
foreach coverTest ∈  examples{
```

Zunächst muss jedes Beispiel der Trainingsmenge angeschaut werden. Genau genommen wird der nachfolgende Programm-Code so oft durchgegangen, wie es in der Trainingsmenge Beispiele gibt. In jedem Durchlauf wird in *coverTest* das aktuelle Beispiel aus der Trainingsmenge gespeichert. Dann wird getestet ob die aktuelle Regel dieses Beispiel abdeckt, denn nur die Gewichte der Beispiele sollen geändert werden, die auch durch die gelernte Regel abgedeckt wurden. Mit calcNewWeight wird das neue Gewicht des Beispiels *coverTest* berechnet. Mit

```
example.weight = example.weight * MultiValue
```

wird das neue Gewicht des abgedeckten Beispiels auf das alte Gewicht mal den Multiplikationsfaktor gesetzt.
Da *MultiValue* als Default-Wert auf 0.5 gesetzt ist, bewirkt das bei jedem abgedeckten Beispiel ein Halbierung des Gewichtes. Das geschieht mit jeder Regel die dieses Beispiel abdeckt. Zum Schluß wird dann die Anzahl der abgedeckten Regeln für dieses Beispiel um eins erhöht. Dies geschieht mit `coverTest.covered++`. Die Anzahl der Regeln die ein Beispiel abdecken, sind in diesem Gewichtungsmodell noch unrelevant, könnten aber für ein anderes Gewichtungsmodell von Interesse sein.
Wenn die Schleife alle Beispiele der Trainingsmenge nach Treffern durchsucht hat, werden die Beispiele an die aufrufende Methode (separateAndConquer von AbstractSeco) zurückgegeben.

5.4.3 Implementierung des komplexen Gewichtungsmodell

In Quelltext 5.7 ist der Pseudocode der Lightweight-Methode aufgezeigt, die in 5.2.7 beschrieben wurde.

Quelltext 5.7 Komplexes Gewichtungsmodell WeightNegativeToThePower

```
procedure changeWeights(examples, rule){
  foreach coverTest ∈  examples{
    if(rule.covers(coverTest) AND coverTest.isNegative){
      newweight = calcNewWeight(coverTest)
      coverTest.weight = newweight
      coverTest.covered++
    }
  }
  return examples
}

procedure calcNewWeight(example){
  weight = example.covered^MultiValue
  weight = weight + AddValue
  return weight
}
```

Wie auch schon in 5.6 wird mit

```
foreach coverTest ∈  examples{
```

der nachfolgende Schleifen-Körper so oft durchlaufen, wie es Beispiele in der Trainingsmenge gibt. Auch hier wird das aktuelle Beispiel in *coverTest* gespeichert. Dann folgt ein Test, ob die übergebene Regel das coverTest abdeckt und ob *coverTest* ein Negativ-Beispiel ist. Mit

```
weight = example.covered^MultiValue
```

wird das neue Gewicht berechnet, wie in der folgenden Formel

$$newweight = coveredCount^{MultiValue}$$

angegeben. Wie auch schon bei dem einfachen Modell wird jetzt das Gewicht des Beispiels auf den neu errechneten Wert gesetzt. Um die Berechnung von der Zuweisung zu trennen wurde zur Berechnung des neuen Gewichtes die Methode calcNewWeight eingeführt. Danach wird die Variable, die die Anzahl an abgedeckten Regeln speichert, um eins erhöht. Nachdem alle Beispiele durchgetestet wurden, werden alle Beispiele mit neuen Gewichten zurückgegeben.

5.5 Konfiguration für Weighted Covering

Um das Weighted Covering nun nutzen zu können, müssen Parameter in der XML Konfigurationsdatei geändert und neue hinzugefügt werden. Das *Rule Stop Criterion* muss geändert werden.Durch die Änderung soll der Lernalgorithmus feststellen können, ob ausreichend Regeln gelernt wurden. Das geänderte rulestopcriterion wird wie in Quelltext 5.8 eingtragen.

Quelltext 5.8 AQR mit Covering (Ausschnitt)

```
<secomp interface="rulestopcriterion" classname="AqrRuleStopWeighted">
  <property name="positveHasToStop" value="1.0"/>
</secomp>
```

Als nächstes muss noch das Gewichtungskriterium für die Beispiele angegeben werden. Da dies ein eigenständiges Interface ist, wird kein Parameter geändert, sondern ein neuer Interface-Abschnitt hinzugefügt. Das Ergebnis für das einfache Gewichtungsmodell aus 5.2.6 findet sich in Quelltext 5.9.

Quelltext 5.9 Einfaches Gewichtsmodell in XML

```
<secomp interface="weightcriterion" classname="WeightPerIteration">
  <property name="MultiValue" value="0.5"/>
</secomp>
```

Als letztes muss der SeCo-Factory noch mitgeteilt werden, dass Weighted Covering genutzt werden soll. Dies geschieht mit dem folgenden Parameter:

Quelltext 5.10 Paramter der Factory

```
...
<property name="weighted" value="true"/>
...
```

Eine komplette XML Konfigurationsdatei für den AQR-Algorithmus ist in Quelltext 5.11 zu sehen.

5.6 Neue Variable Funktionen der SeCo

Die variablen Funktionen der SeCo-Factory wurden durch die Möglichkeit des Weighted Coverings erweitert. Die Tabelle der variablen Funktionen aus 4.1 wurde erweitert zur Tabelle 5.2.

Tabelle 5.2: Variable Funktionen eines SeCo-Algorithmus

Funktion	Beschreibung
RuleStoppingCriterion	Das Kriterium bestimmt, wann der Lernprozess endet.
PostProcess	Ermöglicht die Nachbearbeitung der Regeln.
InitializeRule	Bestimmt die erste Regel, mit der die Verfeinerung begonnen wird.
EvaluateRule	Bewertet Regeln mit Hilfe einer Heuristik.
SelectCandidates	Bestimmt die nächsten Kandidaten der Regeln, die verfeinert werden.
RefineRule	Verfeinert eine Regel und gibt alle Verfeinerungen zurück.
StoppingCriterion	Entscheidet mit Hilfe einer Heuristik, ob eine weitere Verfeinerung der Regel sinnvoll ist.
FilterRules	Ermöglicht es, die Menge der Regeln nach gewissen Kriterien zu filtern.
WeightCriterion	Ermöglicht die Gewichtung von Beispielen

Als Element der SeCo XML-Beschreibung kam das weightcriterion hinzu. Somit ändert sich die Tabelle der Elemente der SeCo XML-Beschreibung 4.5 zur Tabelle 5.3.

Quelltext 5.11 AQR, gewichtete XML

```
<seco>
  <secomp interface="ruleevaluator" classname="DefaultRuleEvaluator">
      <jobject package="seco.heuristics" classname="AqrDifference"
        setter="heuristic"/>
   </secomp>

  <secomp interface="rulerefiner" classname="AqrRefinerTopDown"
    package="seco.learners.aqr">
    <property name="maxStar" value="30"/>
    <property name="negativeSelection" value="0"/>
    <property name="seedChoice" value="random"/>
  </secomp>

  <secomp interface="selector" classname="DefaultSelector"/>

  <secomp interface="stopcriterion" classname="AqrStopping">
    <property name="threshold" value="0.0"/>
  </secomp>

  <secomp interface="rulestopcriterion" classname="AqrRuleStopWeighted">
    <property name="positveHasToStop" value="1.0"/>
  </secomp>

  <secomp interface="rulefilter" classname="MultiRuleFilter">
    <jobject classname="BeamWidthFilter" setter="filter">
      <property name="beamwidth" value="3"/>
    </jobject>
  </secomp>

  <secomp interface="weightcriterion" classname="WeightPerIteration">
    <property name="MultiValue" value="0.5"/>
  </secomp>

  <property name="weighted" value="true"/>
</seco>
```

Tabelle 5.3: Elemente und Attribute der XML-Beschreibung

Element	**Attribute**	**Beschreibung**
seco		Das Wurzelelement, welches die Konfiguration eines SeCo-Klassifizierers beinhaltet
secomp		Eine SeCo-Komponente
	interface	Die Schnittstelle bzgl. der AbstractSeco-Klasse. Mögliche Werte sind candidateselector, postprocessor, ruleevaluator, rulefilter, ruleinitializer, rulerefiner, rulestoppingcriterion, stoppingcriterion und weightcriterion.
	classname	Name der Java-Klasse, welche die Komponente implementiert.
	package	(Optional) Das Java-Package, welches die angegebene Java-Klasse beeinhaltet. Als Standard wird seco.learners.components angenommen.
jobject		Ein beliebiges Java-Objekt
	classname	siehe secomp
	package	siehe secomp
	setter	Teilname der setter-Methode, welche für Aggregation dieses Objektes mit dem Objekt der nächsthöheren Ebene verwendet wird. Wenn also die Java-Methode z.B. setHeuristic heißt, so muss als setter 'heuristic' angegeben werden.
property		Zur Festlegung einer spezifischen Eigenschaft eines Objektes.
	name	Name der Eigenschaft
	value	Wert der Eigenschaft

6 Evaluierung der Algorithmen und der Varianten

In diesem Kapitel gehen wir darauf ein, was Evaluierung ist und wie in dieser Arbeit evaluiert wird. Anschließend erfolgt die Evaluierung an verschiedenen Datensets und die Präsentation der Ergebnisse.

6.1 Was ist Evaluation?

Evaluation steht für Auswertung / Bewertung. Wir wollen verschiedene Algorithmen mit verschiedenen Parametern vergleichen. Durch die Auswertung kann man erkennen, welche Algorithmen sich unter welchen Parametrisierungen für welchen Daten eignen und welche nicht.
Wir bewerten die Algorithmen aufgrund ihres Abschneidens bei vorher festgelegten Datensets. Diese Datensets entstammen alle dem UCI-Repository [23]. Die Daten wurden im ARFF-Format direkt aus [24] bezogen. In Tabelle 6.2 gibt es eine Auflistung der verwendeten Lernprobleme und in Tabelle 6.1 eine Erklärung zu den Tabellenspalten von 6.2.

Tabelle 6.1: Legende der Eigenschaften der Lernprobleme

Spalte	**Beschreibung**
Problem	Name des Lernproblems
Attr.	Gesamtanzahl der Attribute
Nom.	Anzahl an nominalen Attributen
Num.	Anzahl an numerischen Attributen
Kl.	Gesamtanzahl der verschiedenen Klassen
Beisp.	Anzahl der vorhandenen Beispiele
größte Kl.	Prozentsatz des Vorkommens der größten Klasse

In Tabelle 6.2 kommen 8 Datensets ohne numerische Attribute aus:
audiology, breast-cancer, kr-vs-kp, mushroom, primary-tumor, soybean, splice, vote.
Sechs der Datensets kommen mit wenigen numerischen Attributen aus:
anneal, anneal.ORIG, hepatitis, lymph, sick, zoo.
20 Datensets haben viele numerische Attribute:
autos, balance-scale, breast-w, colic, colic.ORIG, credit-a, credit-g, diabetes, glass, heart-c, heart-h, heart-statlog, hypothyroid, ionosphere, iris, labor, segment, sonar, vehicle, vowel.
Dies ist insbesonders wichtig zu wissen, falls ein Algorithmus nicht gut mit numerischen Daten umgehen kann. Ob dies der Fall ist, darauf wird später nochmal eingegangen.

Tabelle 6.2: Verwendete Lernprobleme des UCI

Problem	**Attr.**	**Nom.**	**Num.**	**Kl.**	**Beisp.**	**größte Kl.**
anneal	38	32	6	6	898	76,00%
anneal.ORIG	38	32	6	6	898	76,00%
audilogy	69	69	0	24	226	25,00%
autos	25	10	15	7	205	33,00%
balance-scale	4	0	4	3	625	46,00%
breast-cancer	9	9	0	2	286	70,00%
breast-w	9	9	0	2	699	66,00%
colic	22	15	7	2	368	63,00%
colic.ORIG	27	20	7	2	368	63,00%
credit-a	15	9	6	2	692	55,00%
credit-g	20	13	7	2	1000	70,00%
diabetes	8	0	8	2	768	65,00%
glass	9	0	9	7	214	36,00%
heart-c	13	7	6	5	303	54,00%
heart-h	13	7	6	5	294	54,00%
heart-statlog	13	0	13	2	270	56,00%
hepatitis	19	13	6	2	155	79,00%
hypothyroid	29	22	7	4	3772	92,00%
ionosphere	34	0	34	2	352	64,00%
iris	4	0	4	3	150	33,00%
kr-vs-kp	36	36	0	2	3196	52,00%
labor	16	8	8	2	57	65,00%
lymph	18	15	3	4	148	55,00%
mushroom	22	22	0	2	8124	52,00%
primary-tumor	17	17	0	22	339	25,00%
segment	19	0	19	7	2310	14,00%
sick	29	22	7	2	3772	94,00%
sonar	60	0	60	2	208	53,00%
soybean	35	35	0	19	684	13,00%
splice	61	61	0	3	3190	52,00%
vehicle	18	0	18	4	846	26,00%
vote	16	16	0	2	435	61,00%
vowel	13	3	10	11	990	09,00%
zoo	17	17	0	7	101	41,00%

6.2 Evaluationsmaße

In Kapitel 2.4 haben wir bereits einige Kriterien für die Vergleichbarkeit von Lern-Algorithmen kennen gelernt (z.B. die Korrektheit, die Größe der Regelmenge und die Komplexität).

6.2.1 Korrektheit

Es gibt zwei Möglichkeiten die Korrektheit zu messen: durch Messung auf den Trainingsdaten und durch Messung auf speziellen Testdaten.

Messung auf Trainingsdaten Bei der Messung auf Trainingsdaten werden die Daten, die benutzt wurden um die Theorie zu finden, genutzt, um die Korrektheit zu messen. Korrektheit besagt hier, wie viele Beispiele eine korrekte Zuordnung, durch die Regeln, erhalten haben. Eine Korrektheit von 100% würde bedeuten, dass alle Beispiele durch die Regeln richtig klassifiziert wurden. Dabei kann man sehen, ob die Theorie die gelernten Beispiele alle korrekt klassifiziert. Viele Algorithmen treffen die Annahme, dass hier die Korrektheit bei 100% liegen muss. Aber Regellernalgorithmen erreichen hier nur selten eine Korrektheit von 100%. Da ja beliebig viele Regeln gelernt werden können, kann sich ein Algorithmus auch *perfekt* an das Trainings-Datenset anpassen. Das ist aber oft gar nicht gewollt. Durch diese große Anpassung an die Trainingsdaten entsteht sogenanntes **Overfitting**, d.h. die Regeln sind viel spezieller als das eigentlich gesuchte Konzept. Dieser Effekt ist vor allem dann von Nachteil, wenn ein Teil der Daten fehlerhaft ist. Dann werden fehlerhafte Regeln gelernt und spätere Beispiele falsch eingeordnet.

Messung mit Testdaten Bei der Messung mit Testdaten wird meist ein Teil der Trainingsdaten zurückgehalten. Diese werden dann später genutzt, um die Wirksamkeit der Theorie auf „neuen“ Daten zu testen. Der Nachteil dieser Methode ist offensichtlich: Dem Lern-Algorithmus stehen weniger Beispiele in der Trainingsmenge zur Verfügung. Dies ist vor allem ein Nachteil im Kostenfaktor, denn der Aufwand für das Klassifizieren von Daten ist nicht unerheblich. Um dieses Problem zu umgehen wird meist eine *10-fold cross validation* durchgeführt, wie sie in der Weka-Bibliothek implementiert ist. Die Trainingsmenge wird dabei in 10 gleich große Mengen aufgeteilt. Nun bekommt der Lern-Algorithmus 9 Teile zum Lernen und der 10te dient zum Messen des Lernerfolgs. Dies wird 10 mal durchgeführt, so dass jede Menge ein Mal als Testmenge genutzt wird. Die Ergebnisse dieser 10 Messungen werden dann durch das arithmetische Mittel zusammengefasst. Dieses wird als eine Abschätzung für die Genauigkeit der Theorie genutzt, die aus allen Beispielen gelernt wird. So stehen dem Lern-Algorithmus

alle Beispiele zum Lernen zur Verfügung und es gibt dennoch eine Aussage über die Korrektheit dieser Theorie auf ungesehenen Daten.

6.2.2 Größe der Regelmenge

Eine weitere Information, die uns zur Verfügung steht, ist die Größe der Regelmenge. Bei den verwendeten Lern-Algorithmen ist zu beachten, dass für **jede** Klasse Regeln gelernt werden, obwohl es möglich wäre, die Regeln für die letzte Klasse wegzulassen und diese als Default-Klasse festzulegen. Wir definieren die Größe der Regelmenge als Anzahl der Regeln in der erlernten Regelmenge.

6.2.3 Anzahl der Bedingungen

Ergebnisse nur anhand der Anzahl an Regeln zu vergleichen, ist nicht sinnvoll. Es gilt noch zu berücksichtigen, wie viele Bedingungen in der Regelmenge vorkommen. Somit werden kürzere Regeln den längeren Regeln gegenüber bevorzugt. Wir zählen die Summe alle Bedingungen, die in der Regelmenge auftauchen.

6.3 Vergleich von AQR und CN2

Die CN2 Variante, die hier Verwendung findet, baut auf der BEXA-Implementierung von Mathias Thiel auf [1]. Für AQR wurden folgende Paramter genutzt:
AqrDifference als Heuristik, *negativeSelection = 0* (Abstand), *seedChoice = random*, *stopcriterion = Aqrstopping* mit dem *threshold von 0.0*, *rulestopcriterion = AqrRuleStop* sowie als *rulefilter* wurde *BeamWidthFilter* mit einem *beamwidth = 3* genutzt.
Für CN2 wurden folgenden Paramter genutzt:
Laplace wurde als Heuristik und als *stopcriterion* die *LikelihoodRatio* mit einem *threshold = 0.9* genutzt. Als Regelfilter kamen ein *ChiSquareFilter* mit einem *threshold = 0.9* und ein *BeamWidthFilter* mit einer beamwidth = 3 zum Einsatz.

Als Vergleich wurde auch eine Mischform betrachtet: CN2 mit Seed. In dieser Variante werden, im Gegensatz zur konventionellen Version des CN2, nicht alle Attribute in Betracht gezogen, sondern nur solche, die das Seed-Beispiel abdecken. Diese Variante befindet sich also auf halben Weg von AQ zu CN2.

Um einen Vergleich mit dem State-of-the-Art im Bereich Regel-Lernen zu ermöglichen, wurde der Ripper Algorithmus [6] herangezogen. Verwendet wurde die Implementation JRip aus der Weka-Bibliothek [17]. JRip benutzt ein eingebautes Pruning, weswegen er weniger und kürzere Regeln produziert als AQ und CN2.

Nachfolgend eine Win / Loose Aufstellung der Algorithmen: Insgesamt gab es 34 Testdatenbanken. Es kann vorkommen, dass die Summe aus Win + Loose kleiner

als 34 ist, was daran liegt, dass die Datensets auf denen beide Algorithmen gleich gut waren, nicht gezählt werden.

Tabelle 6.3: Win-Loose Aufstellung AQR und CN2

	AQR	**CN2**	**CN2+Seed**	**JRip**
AQR	-/-	12/20	12/21	8/26
CN2	20/12	-/-	15/17	8/24
CN2+Seed	21/12	17/15	-/-	12/20
JRip	26/8	24/7	20/12	-/-

In dieser Win + Loose Aufstellung ist sehr gut zu sehen, dass der AQ gegen keinen anderen Algorithmus gut abschneidet. Eine Vermutung ist, dass der AQR sich zu sehr an die Beispiele anpasst, also Overfitting auftritt. Hier könnte vielleicht Pruning helfen. Anderseits könnte die Klassifikation ungeeignet sein. Vielleicht wäre eine Klassifikation besser geeignet, die eine Mehrheitsentscheidung der Regeln trifft.

6.3.1 Ergebnisse des Vergleichs von AQR und CN2

Der AQR ist im Schnitt 0,56 Prozentpunkte schlechter als der CN2-Algorithmus. Aber die maximale Verbesserung, die AQR erreichte, lag bei 35,95%. Während CN2 im besten Fall fast 50 Prozentpunkte besser als AQR klassifizierte. Der CN2 lernt mehr Regeln mit mehr Bedingungen als der AQR. Interessant ist hier, dass CN2+Seed im schlimmsten Falle 24,04 Prozentpunkte (*vowel.arff*) schlechter war als AQ, aber nur 4,24 (vowel.arff) Prozentpunkte schlechter als CN2. Im besten Fall war CN2+Seed 48,96 Prozentpunkte (balance-scale.arff) besser als AQR und 35,42 Prozentpunkte (splice.arff) besser als CN2. JRip war im Schnitt 4 Prozentpunkte besser als die anderen Algorithmen. Der CN2 war auf dem Datenset *labor.arff* 10,53 Prozentpunkte besser als JRip. Das ist ein sehr guter Wert.

Wie in Tabelle C.4 ersichtlich, hat der AQ-Algorithmus ein Problem mit den Datensets balance-scale.arff und primary-tumor.arff, auf denen er besonders schlecht abschneidet. Das Datenset balance-scale hat 4 numerische Attribute und 0 nominale Attribute. Dies legt die Vermutung nahe, dass entweder der AQ-Algorithmus oder dessen Implementierung ein Problem mit numerischen Daten haben könnte. Aber bei genauerer Analyse der Datensets müsste das bedeuten, dass AQ auf allen Datensets mit numerischen Daten schlecht abschneidet. Dies ist jedoch nicht der Fall. So schneidet der AQ-Algorithmus z.B. bei dem vowel.arff Datenset, welches aus 10 numerischen und 3 nominalen Attributen besteht, sogar besser ab als der CN2-Algorithmus. Auch auf anderen Datensets, die nur aus numerischen Attributen bestehen, wie z.B. sonar.arff, schneidet der AQ-Algorithmus sehr gut ab. Um den Einfluß von numerischen Attributen auszuschließen, wurden zu Testzwecken

im Datenset balance-scale.arff die numerischen Attributewerte 1-5 ersetzt durch Attributewerte a-e. Trotz dieser Diskretisierung schnitt der Algorithmus mit einer Korrektheit von 53,60% immer noch sehr schlecht ab. Bei der Betrachtung der Regelmengen ergibt sich ein erstaunliches Bild: Der AQ lernt 81 Regeln mit insgesamt 576 Bedingungen, während der CN2 nur 51 Regeln mit 166 Bedingungen lernt. Es liegt also nahe, dass der AQ ein Problem mit Overfitting auf diesem Datenset hat.

Der Einfluß der verschieden Parameter der Algorithmen wurde in Tabelle C.9 für AQR und in Tabelle C.10 für CN2 untersucht. Es wurden verschiedene Parameter variiert und die Ergebnisse mit den Original-Algorithmen verglichen. Für AQR war ein Parameter signifikant besser als die Original-Parameter: Mit einer *Beamwidth = 30* gewann AQR auf 23 Datensets und verlor gegen die *Beamwidth = 3* auf 10 Datensets.
Bei dem CN2 Algorithmus läßt sich nicht so leicht ein guter Parameter finden. Nur der Parameter *Beamwidth = 1* war der Einstellung *Beamwidth = 3* in 21 Fällen überlegen und in 10 Fällen unterlegen. Dies ist an der Grenze zur Signifikanz.
Im Anhang C finden sich genaue Tabellen mit dem Abschneiden der Algorithmen auf den Datensets, während in den Tabellen C.6 und C.7 sich dafür die benötigten Regeln und Bedingungen befinden.

6.4 Einfluß von Weighted Covering

In diesem Kapitel wollen wir uns den Einfluß des Weighted Covering auf die verschiedenen Algorithmen näher betrachten: zunächst der AQR-Algorithmus, gefolgt von dem CN2-Algorithmus und dann schließlich die CN2 mit Seed Variante. Bei allen Algorithmen, bei denen Weighted Covering Verwendung gefunden hat, wurde der Gewichtungsfaktor 0.5 genutzt, d.h. das Gewicht aller abgedeckten Beispiele wurde halbiert.

6.4.1 AQR gegen AQR+W.C.

In Tabelle C.2 folgt eine direkte Gegenüberstellung der AQR Algorithmen ohne und mit Weighted Covering, während in Tabelle 6.4 eine direkte Win / Loose Aufstellung folgt.

Tabelle 6.4: Win / Loose Aufstellung AQR und AQR+W.C.

	AQR	**AQR+W.C.**	**AQR+Corr**	**AQR+W.C.+Corr**
AQR	-/-	27/6	19/13	25/7
AQR+W.C.	6/27	-/-	9/22	21/13
AQR+Corr	13/19	22/9	-/-	22/8
AQR+W.C.+Corr	7/25	13/21	8/22	-/-

Klar ersichtlich ist, dass der AQ-Algorithmus ohne Weighted Covering Signifikant besser ist als mit Weighted Covering. Nur in sechs Fällen hat das Weighted Covering hier Vorteile erbracht.

Auch hier wurden der Einfluß der verschiedenen Parameter untersucht. In Tabelle C.11 finden sich die Ergebnisse für AQR mit Weighted Covering und in Tabelle C.10 die Ergebnisse für CN2 mit Weighted Covering. Für AQR mit Weighted Covering war der beste Parameter *Beamwidth = 40*, damit gab es Verbesserungen bis zu 14,05 Prozentpunkte (*credit-g.arff*), aber auch Verschlechterungen um bis zu 7,35 Prozentpunkte (*vote.arff*). Im Schnitt ergab sich noch eine Steigerung um 1,4 Prozentpunkte.
Bei CN2 mit Weighted Covering gab es auch wie bei CN2 keinen signifikant besseren Parameter. Nur der Parameter *weight = 0.25* war auf 20 Datensets etwas besser als *weight = 0.5*. Der Parameter *weight = 0.25* ist knapp an der Grenze zur Signifikanz.
Bei der Betrachtung der Regelmengen und der Anzahl an Bedingungen in C.6 und C.7 fällt auf, dass das Weighted Covering bei AQR zu einem starken Anstieg an Regeln und Bedingungen geführt hat. Hier wäre der Einfluß von Pruning sehr interessant, da scheinbar AQR mit Weigted Covering zu starkem Overfitting neigt. Nur bei wenigen Datensets ist die AQR + Weighted Covering Variante der AQR Variante überlegen. Auch das auswechseln der Bewertungsheuristik mit *Correlation* hat keine signifikante Verbesserung erbracht. Noch immer ist der AQR Algorithmus besser als die AQR Weighted Covering Variante.

6.4.2 CN2 gegen CN2+W.C.

Hier betrachten wir den CN2-Algorithmus und die Folgen des Weighted Covering. In Tabelle 6.5 folgt eine direkte Win / Loose Aufstellung des CN2 und des CN2 mit Weighted Covering.

Tabelle 6.5: Win / Loose CN2 und CN2+W.C.

	CN2	**CN2+W.C.**
CN2	-/-	10/22
CN2+W.C.	22/10	-/-

Die Weighted Covering Variante des CN2 ist auf 22 Testdatenbanken besser als der Standard CN2 Algorithmus und nur auf 10 schlechter. Das ist an der Grenze

zur Signifikanz. Die Genauigkeit steigt durch den Einsatz von Weighted Covering auf dem Datenset *splice.arff* um bis zu 39,53 Prozentpunkte. Kann aber auch um 21,83 Prozentpunkte sinken, wie bei dem Datenset *anneal.ORIG*. Im Schnitt bessert sich die Genauigkeit um 1,72 Prozentpunkte. Hier lohnt sich der Einsatz des Weighted Covering für CN2. Bei Betrachtung der Klassifikationsleistung liefert der CN2 mit Weighted Covering auf sieben Datensets die besten Leistungen im Vergleich zu den anderen Algorithmen (mit Ausnahme von JRip). Die Regeln, die durch Weighted Covering gelernt wurden, verdoppeln fast die Anzahl der Bedingungen. Auch hier wäre ein Pruning notwendig. Aber da auch einige Regeln mehr gelernt wurden, wäre eine andere Klassifikationsmethode hier vielleicht sinnvoll.

Tabellen mit den erzielten Ergebnissen der Korrektheit finden sich in Tabelle C.4. In den Tabellen C.6 und C.7 finden sich die dafür benötigten Regeln und Bedingungen. Wie bei CN2 konnte bei CN2 mit Weighted Covering kein Parameter signifikant die Leistung der Original Parameter verbessern. Der Parameter *weigth=0.25* zeigt eine Steigerung um maximal 5,12 Prozentpunkte (*anneal.ORIG*), aber dafür eine Verschlechterung um bis zu 2,02 Prozentpunkte(*lymph.arff*). Im Schnitt verbesserte sich die Leistung nur um 0.78 Prozentpunkte.

6.4.3 CN2+Seed gegen CN2+Seed+W.C.

Dieser Abschnitt ist der Mischform aus AQR und CN2 gewidmet. Uns interessiert die erzielten Ergebnisse des CN2+Seed mit Weighted Covering im Vergleich zum CN2+Seed. In Tabelle 6.6 findet sich eine Gegenüberstellung des Algorithmus.

Tabelle 6.6: Win / Loose CN2+Seed und CN2+Seed+W.C.

	CN2+Seed	**CN2+Seed+W.C.**
CN2+Seed	-/-	18/14
CN2+Seed+W.C.	14/18	-/-

Auch hier zeigt sich das Bild, das die Weighted Covering Algorithmen auf dem Datenset *vowel.arff* sehr gut abschneiden und hier eine Steigerung der Klassifikation ermöglicht. Die Klassifikationsleistung bei *vowel.arff* steigert sich um 10,60 Prozentpunkte. Verschlechtert hat sich die Klassifikation am größten bei *anneal.ORIG* um 14 Prozentpunkte.
Das schlechte Abschneiden des Weighted Coverings könnte an dem vorgehen der Klassifizierung der SeCo-Factory liegen. Da einfach die erste passende Regel ein unbekanntes Beispiel klassifiziert, hat das Weighted Covering nicht die Chance die gelernte Redundanz zum Tragen kommen zu lassen. Die Redundanz ist der Hauptvorteil des Weighted Covering, aber um diese nutzen zu können, muss die Klassifikation der SeCo-Factory geändert werden. Dann ist mit deutlich besseren Ergebnissen zu Rechnen.

Genauere Auflistungen der Korrektheit finden sich in Tabelle C.8, während sich in Tabelle C.6 und C.7 Auflistungen über die Anzahl an Regeln und Bedingungen der einzelnen Algorithmen und Datensets finden.

6.5 Win-Loose Aufstellung

Zur besseren Übersicht über die Performance der Algorithmen wurde eine Win-Loose Aufstellung erstellt. Diese findet sich in Tabelle 6.7. Dort findet sich auch eine spezielle Variante des CN2: CN2 mit Seed. In dieser Variante werden, im Gegensatz zur konventionellen Version des CN2, nicht alle Attribute in Betracht gezogen, sondern nur solche, die das Seed-Beispiel abdecken. Diese Variante befindet sich also auf halben Weg von AQ zu CN2.
In dieser Win + Loose Aufstellung ist sehr gut zu sehen, dass der AQ gegen keinen anderen Algorithmus gut abschneidet. Nur gegen die Weighted Covering Variante von AQ gewinnt der Standard AQ signifikant. Eine Vermutung ist, dass die Klassifikation ungeeignet ist. Vielleicht wäre eine Klassifikation besser geeignet, die eine Mehrheitsentscheidung der Regeln trifft.
Der CN2 schlägt sich ganz gut, ist aber nie signifikant besser als einer der anderen Algorithmen Varianten. Die Weighted Covering Variante des CN2 ist auf 22 Testdatenbanken besser als der Standard CN2 Algorithmus und nur auf 10 schlechter. Das ist an der Grenze zur Signifikanz.
Die Bexa-Implementierung der SeCo-Factory ist nur dem AQ Algorithmus überlegen, ansonsten den anderen Algorithmen unterlegen.
Die spezielle CN2+Seed Variante ist stark signifikant besser als der AQ mit Weighted Covering, ansonsten weit davon entfernt gegen andere Algorithmen signifikant besser zu sein. Aber die CN2+Seed Variante schlägt sich ausgesprochen gut gegen Bexa und AQ.

6.6 Vergleich alle Algorithmen

Hier findet sich ein kurzes Resümee über alle Algorithmen und deren Eigenschaften. CN2 und Bexa schneiden ähnlich ab. Das ist nicht überraschend, da die hier genutzte Variante von CN2 ein Bexa mit CN2-Verhalten ist.
Die Ergebnisse von CN2 mit Seed und AQ sind sich nicht ähnlich, auch wenn beide, als Besonderheit, die Verfeinerung mit Hilfe eines Seeds nutzen. Im schlechtesten Fall war der CN2 mit Weighted Covering um 21 Prozentpunkte schlechter als der originale CN2-Algorithmus. Dafür war aber im besten Fall eine Steigerung um fast 40 Prozentpunkte möglich, z.B. bei dem Datenset splice.arff: CN2=52,73% zu CN2+W.C.=92.26%.
Algorithmen mit Weighted Covering brauchen teilweise bedeutend mehr Regeln und Attribute als Algorithmen ohne Weighted Covering.
Im Schnitt unterscheiden sich die Algorithmen mit und ohne Weighted Covering

um 3 bis 5 Prozentpunkte. Die maximalen Genauigkeitsunterschiede lagen zwischen 10 und fast 40 Prozentpunkte. In nur sehr wenigen Fällen wurde durch Weighted Covering weniger Regeln und Bedingungen gelernt als durch die Algorithmen ohne Weighted Covering. Meist wurden zwischen wenigen und ca. 162 Regeln mehr gelernt. Auch die Anzahl an den verwendeten Bedingungen ist teilweise stark gestiegen: von wenige Bedingungen mehr, bis hin zu knapp 900 Bedingungen mehr, als die gleichen Algorithmen ohne Weighted Covering.

Bei 19 Datensets hatte der beste Algorithmus weniger Regeln und Bedingungen gelernt als der Schlechteste, und bei 12 Datensets hatte der beste Algorithmus mehr Regeln gelernt als der schlechteste. Bei drei Datensets wahren entweder die Anzahl der Regeln oder die Anzahl der Bedingungen gleich.
Deswegen erfolgt in Tabelle 6.7 zusätzlich ein Überblick über alle Algorithmen in einer Win / Loose Tabelle.

6.6.1 Einfluss der Klassifikation

Im folgenden soll noch erwähnt werden, dass noch versucht wurde, den Einfluss der Klassifikation der SeCo-Factory zu untersuchen. Genauer wurde die Klassifikation nach der ersten Feuernden Regel versuchsweise ersetzt durch die Klassifikation der Mehrheit der Regeln. Also es wurde gezählt wie viele Regeln ein Beispiel einer gewissen Klasse zuordneten. Da das für alle Klassen gemacht wurde, bekam das Beispiel am Ende die Klasse, mit den meisten klassifizierenden Regeln.

AQ und W.C. Die Weighted Covering Variante des AQ-Algorithmus hat sehr stark von der geänderten Klassifizierung profitiert. Auf 19 Datensets war der AQ W.C. mit Mehrheitsklassifikation der Standard AQ mit W.C. Variante überlegen. Die Klassifikationsleistung steigt im Schnitt um 1,48 Prozentpunkte an. Die beste Steigerung ließ sich bei *zoo.arff* erreichen: 24.71 Prozentpunkte. Während die schlechteste Leistung auf dem Datenset *soybean.arff* zu finden war: eine Verschlechterung um 12.88 Prozentpunkte. Aber das schlechte Abschneiden des AQ-Algorithmus mit Weighted Covering konnte auch die geänderte Klassifikation nicht ändern. Das legt die Vermutung nahe, dass die Klassifikation nicht das Hauptproblem darstellt. Sondern eher das extreme Overfitting des Algorithmus. Betrachtet man sich die Regelmengen der anderen Algorithmen scheint sich dies zu bestätigen.

CN2 und W.C. Die Weighted Covering Variante des CN2-Algorithmus setzt sich sehr gut gegen einige Varianten der anderen Algorithmen durch. Die Klassifikationsleistung ist signifikant besser als die des AQ-Algorithmus mit Weighted Covering und Mehrheitsklassifizierung. Die Leistung ist sogar signifikant besser als der Orginal CN2-Algorithmus. Es ließ sich eine Steigerung um bis zu 39.5

Tabelle 6.7: Win-Loose Aufstellung

Win/Loose	CN2+Seed	Bexa	CN2	AQ	AQ+W.C.	JRip	Bexa+W.C.	CN2+W.C.	CN2+S+W.C.	Summe
CN2+Seed	-/-	19/12	17/15	21/12	25/9	12/20	12/20	13/20	18/14	137/122
Bexa	12/19	-/-	15/15	21/12	23/11	9/23	11/21	13/19	18/16	122/136
CN2	15/17	15/15	-/-	20/12	23/11	8/24	9/22	10/22	17/16	116/139
AQ	12/21	12/21	12/20	-/-	27/6	8/26	8/25	10/22	17/17	107/158
AQ+W.C.	9/25	11/23	11/23	6/27	-/-	7/27	4/29	7/26	10/24	65/204
JRip	20/12	23/9	24/7	26/8	27/7	-/-	23/8	24/10	25/7	192/68
Bexa+W.C.	20/12	21/11	22/9	25/8	29/4	8/23	-/-	17/13	22/12	164/92
CN2+W.C.	20/13	19/13	22/10	22/12	26/7	10/24	13/17	-/-	20/12	152/108
CN2+S+W.C.	14/18	16/18	16/17	17/16	24/10	7/25	12/22	12/20	-/-	118/146
Summe vs.	122/137	136/122	139/116	158/107	204/65	68/192	92/164	108/152	146/118	-/-

Prozentpunkte erreichen (*splice.arff*), einhergehend mit sinkender Leistung bei folgenden Datensets: *sick.arff, anneal.ORIG.arff*. Im Schnitt steigerte sich die Klassifikationsleistung um 2.11 Prozentpunkte.

CN2 mit Seed und W.C. Auch diese Weighted Covering Variante profitiert von der geänderten Klassifikation. Gegen die Standard-Klassifikation gewinnt die Mehrheitsklassifikation des CN2 mit Seed und W.C. auf 21 Datensets. Bei CN2 mit Seed und Weighted Covering steigert sich die Klassifikationsleistung nur um maximal 6.08 Prozentpunkte. Im Schnitt steigert sich die Klassifikation um 0.7 Prozentpunkte. Im Vergleich zu der CN2 mit Seed-Variante ohne Weighted Covering steigert sich die Leistung auf maximal 12.62 Prozentpunkte, verschlechtert sich aber auch um maximal 13.58 Prozentpunkte. So das die Klassifikationsleistung, im Schnitt, nur um 0.15 Prozentpunkte steigt.
Die schlechten Leistungen lassen sich am besten mit Overfitting erklären. Das wäre ein Punkt der noch untersucht werden sollte.
Die genauen Leistungen der Algorithmen lassen sich in den Tabellen C.14 und C.15 im Anhang C nachvollziehen.

7 Schlußwort

7.1 Zusammenfassung

Begonnen wurde in Kapitel 2 mit einer Beschreibung des **Lernproblems** und die Lösung dieses Problems mit Hilfe von **Separate-and-Conquer Algorithmen**. Dabei wurde der Separate-and-Conquer Algorithmus mit Hilfe von Pseudo-Code beschrieben. In Kapitel 3 wurden die, in dieser Diplomarbeit verwendeten, Algorithmen **AQ** und **CN2** vorgestellt. In Kapitel 3.1.1 wird dann auf die Besonderheit des AQ Algorithmus eingegangen, die Nutzung eines Seeds zur Verkleinerung des Suchraums. Beide Algorithmen wurden anhand von Pseudo-Code erklärt. In Kapitel 4 wurde auf die SeCo-Factory und ihre Funktionen eingegangen. Auch die Konfiguration eines Algorithmus mit Hilfe der SeCo-Factory wurde an einem Beispiel verdeutlicht. Dann wurde in Kapitel 4.3 anhand des AQ Algorithmus gezeigt, wie eigene Algorithmen in die SeCo-Factory implementiert werden können. Schließlich wurde in Kapitel 5 auf die Möglichkeit des Weighted Covering eingegangen und die SeCo-Factory wurde um die Möglichkeit erweitert, Weighted Covering für Algorithmen zu nutzen. Dafür wurden notwendige Interfaces definiert und Designvorschläge für eigene Algorithmen festgehalten. Schließlich wurde in Kapitel 6 festgehalten, dass der AQ Algorithmus und das Weighted Covering zwar grundsätzlich funktionieren, aber im Vergleich zu CN2 schlecht abschneiden. Auch wurde der CN2 und der AQR auf 34 Datensets der URI gegeneinander getestet, die mit verschiedenen Parametern und auch unter dem Einsatz des Weighted Covering gegeneinander antraten. Es wurde außerdem eine Mischform von AQR und CN2 implementiert, CN2 mit Seed, die sich auf dem Wege von AQR zu CN2 befindet. Zusätzlich wurden die Algorithmen mit dem State-of-the-Art Regellerner JRip aus der Weka-Bibliothek verglichen.

7.2 Schlußfolgerungen

Es zeigt sich, dass der sehr einfache Ansatz des AQR gegen die neueren Algorithmen keinerlei Chancen hat. Auch der Einsatz von Weighted Covering bringt dort keine Besserung, ganz im Gegenteil! Durch Weighted Covering verstärkt sich die Overfitting Neigung des AQR noch zusehends und führt zu schlechteren Ergebnissen als ein AQR ohne Weighted Covering. Das macht sich auch in einem großen Anstieg der Anzahl an Regeln und Bedingungen des AQR mit Weighted Covering bemerkbar. Der CN2 Algorithmus ist in sehr vielen Fällen besser als der AQR, aber schlechter als JRip. Aber hier steigert sich die Klassifikationsleistung spürbar durch den Einsatz von Weighted Covering. Die Mischform CN2+Seed profitiert nicht so stark von Weighted Covering wie CN2. Nur auf 14 Datensets steigt die Genauigkeit leicht an, auf Kosten von fast einer Verdopplung der Regeln und Bedingungen. Das bedeutet, dass die Nutzung eines so genannten Seeds keine nennenswerte Klassifikationssteigerung mit sich bringt. Die Nutzung von

Weighted Covering bringt eine Steigerung der Korrektheit der Klassifikation mit sich, aber auch die Tendenz zu Overfitting zu neigen. Bei Algorithmen, die von sich aus schon zu Overfitting neigen, wie die Algorithmen die ein Seed nutzen, kann Weighted Covering zu extremen Overfitting führen. Extremes Overfitting führt zu einer deutlichen Senkung der Korrektheit. Nicht auf allen Datensets ist Weighted Covering von Vorteil. Fast alle Algorithmen mit Weighted Covering schnitten z.B. bei dem Pilze Datenset mushroom.arff schlechter ab als die selben Algorithmen ohne Weighted Covering. Auch das Datenset kr-vs-kp.arff war für Weighted Covering nicht geeigntet. Aber es gab auch Datensets auf denen sich Weighted Covering empfiehlt: vowel.arff, labor.arff und audiology.arff.
Weighted Covering bringt zwar teilweise Genauigkeitsanstiege mit sich, jedoch auf Kosten der gelernten Regeln und Bedingungen. Da im Schnitt nur eine Steigerung der Genauigkeit um 4% zu erwarten ist, mit durchschnittlich 70 Regeln mehr, ist Weighted Covering in der Form, in der es in dieser Diplomarbeit genutzt wurde, keine Alternative. Man muss aber beachten, dass das Weighted Covering seinen größten potientiellen Pluspunkt, die Redundanz, nicht nutzen kann, da die SeCo-Factory ungenügende Möglichkeiten vorsieht, die Redundanzinformationen zu nutzen. Dafür würde eine andere Klassifikationsvorgehensweise benötigt.

7.3 Offene Punkte

7.3.1 Algorithmen

Die Algorithmen CN2 und AQ sollten beide noch um ein Pruning ergänzt werden, vor allem die Weighted Covering Varianten lernen ein Vielfaches an Regeln, was mit Pruning vielleicht zu einer Steigerung der Klassifikationsleistung führen kann. Auch eine andere Klassifikation für unbekannte Beispiele, die einen Mehrheitsentscheid trifft, könnte zu einer Steigerung der Weighted Covering Varianten führen.

7.3.2 SeCo-Factory

- Parametrisierbare Klassifikation, da bisher die Klassifikation fest in RuleSet kodiert ist,
- Optimierung des Speicherbedarfs und der Ausführungsgeschwindigkeit,
- Beispielen die Möglichkeit geben, sich zu merken wie viele Regeln deckten dieses Beispiel ab.

Literatur

[1] Thiel, Matthias. Separate and Conquer Framework und disjunktive Regeln. `http://www.ke.informatik.tu-darmstadt.de/lehre/diplomarbeiten/2005/Thiel_Matthias.pdf` TU Darmstadt. 2005

[2] J. Fürnkranz. Separate-and-conquer rule learning. Artificial Intelligence Review. pages 3-54. 1999

[3] Peter Clark and Tim Niblett. The **CN2** induction algorithm. Machine Learning, pages 261-283. 1989

[4] Peter Clark and Robin Boswell. Rule Induction with **CN2**: Some Recent Improvements. Proceedings of the 5th European Working Session on Learning (EWSL-91). pages 151-163. Springer Verlag. 1991

[5] William W. Cohen and Yoram Singer. A Simple, Fast, and Effective Rule Learner. Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99).pages 335-342. AAAI/MIT Press. 1999

[6] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, Proc. of the 12th International Conference on Machine Learning, pages 115-123, Tahoe City, CA, July 9-12, 1995. Morgan Kaufmann.

[7] Sholom M. Weiss and Nitin Indurkhya. Lightweight Rule Induction. Proceedings of the 17th International Conference on Machine Learning (ICML-2000). pages 1135-1142. 2000

[8] Nada Lavrač, Branko Kavšek, Peter Flach, Ljupčo Todorovski. Subgroup Discovery with CN2-SD. Journal of Machine Learning Research, 5(Feb):153–188, 2004.

[9] L.De Raedt and W. Van Laer. Inductive constraint logic. In Proceedings of the 5th Workshop on Algorithmic Learning Theory (ALT-95). Springer, 1995

[10] Hendrik Theron and Ian Cloete. Bexa: A covering algorithm for learning propositional concept descriptions. Journal of Machine Learning, pages 5-40, 1996

[11] J.R. Quinlan. Learning logical definitions from relations, volume 5, 1990

[12] J. Ross Quinlan. Learning efficient classification procedures and their application to chess endgames. In J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, editors, Maschine Learning, vol. 1. Tioga, Palo Alto, Ca, 1983

[13] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI, pages 1137-1145, 1995

[14] R.S. Michalski. On the quasi-minimal solution of the general covering problem. In Proceedings of the 5th international symposium on Information Processing (FCIP 69). Vol. A3 (Switching circuits). Bled, Yugoslavia. pages 125-128. 1969.

[15] Tom Mitchell. Machine Learning. McGraw-Hill. 1997.

[16] Ian Witten & Eibe Frank. Data Mining. Hansa-Verlag. 2001.

[17] Ian H. Witten and Eibe Frank. Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco, 2000.

[18] J. Fürnkranz. Maschinelles Lernen: Symbolische Ansätze. Skript. TU-Darmstadt. 2006.

[19] J. Fürnkranz and Peter A. Flach. ROC 'n' Rule Learning - Towards a Better Understanding of Covering Algorithmus. 2003.

[20] Dirk Schnelle. Kurze Einführung in Log4j. 2005.

[21] Emden Gansner and Eleftherios Koutsofios and Stephen North. Drawing graphs with dot. 2004.

[22] Mihalis Tsoukalos .An Introduction to GraphViz. 2004.

[23] S.Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[24] `http://www.sc.ehu.es/ccwbayes/docencia/mmcc/files/`, `http://www.hakank.org/weka/`, `http://www.ics.uci.edu/~mlearn/MLRepository.html` .

[25] `http://de.wikipedia.org/w/index.php?title=Klasse_(Kategorie)`

A Seco-Factory

A.1 Konfiguration eines SeCo-Algorithmus

Die Beschreibung eines SeCo-Algorithmus findet mit Hilfe von XML statt. Die Beschreibung beinhaltet folgende Attribute und Elemente:

Tabelle A.1: Variable Funktionen eines SeCo-Algorithmus

Funktion	Beschreibung
RuleStoppingCriterion	Das Kriterium bestimmt, wann der Lernprozess endet.
PostProcess	Ermöglicht die Nachbearbeitung der Regeln.
InitializeRule	Bestimmt die erste Regel, mit der die Verfeinerung begonnen wird.
EvaluateRule	Bewertet Regeln mit Hilfe einer Heuristik.
SelectCandidates	Bestimmt die nächsten Kandidaten der Regeln die verfeinert werden.
RefineRule	Verfeinert eine Regel und gibt alle Verfeinerungen zurück.
StoppingCriterion	Entscheidet mit Hilfe einer Heuristik, ob eine weitere Verfeinerung der Regel sinnvoll ist.
FilterRules	Ermöglicht die Menge der Regeln nach gewissen Kriterien zu filtern.
WeightCriterion	Ermöglicht die Gewichtung von Beispielen

A.2 Properties der Objekte

Jedes Objekt der SeCo-Factory bietet die Möglichkeit so genannte *properties* (Eigenschaften) zu definieren. Diese properties geben dem Nutzer die Möglichkeit gewisse Parameter einzustellen. Welche properties es gibt und welche Werte diese properties annehmen können, hängt von dem jeweiligen Objekt ab. In Tabelle A.2 sind ein paar gängige properties angegeben.

Tabelle A.2: Properties verschiedener SeCo-Objekte

Property	**für Element**	**Beschreibung**
threshold	LikelihoodRatio, ChiSquareFilter	Legt den Grenzwert fest, ab welchem Kandidaten Regeln als signifikant betrachtet werden.
beamwidth	BeamWidthFilter	Legt fest, die wie viel besten Kandidatenregeln nach einer Verfeinerung beibehalten werden. Die Übrigen werden verworfen.
nominal.-cmpmode	BexaRefiner	Wenn diese Property nicht gesetzt wird, dann arbeitet der BEXA wie gewöhnlich mit dem $\neq$-Komperator. Bei Angabe von *equal* wird der =-Komparator verwendet und bei *both* sind beide Komparatoren in Gebrauch, was dem BEXA mixed entspricht.
skipdefault-class	AbstractSeco	Diese Einstellung für den Kernel kann auf *true* gesetzt werden, damit für die Standardklasse keine Regel gelernt wird, sondern nur die Standardregel verwendet wird.
weighted	AbstractSeco	Diese Einstellung des Kernel ermöglicht die Verwendung des Weighted Covering für den eigenen Algorithmus. Es muss aber noch die Gewichtungsmethode angegeben werden!
negative-Selection	AqrRefinerTopDown	Legt fest wie die negativen Beispiele ausgewählt werden sollen. Die negativen Beispiele können entweder durch ihre Entfernung zu dem Seed (0), oder zufällig gewählt werden (1). Standard ist auch hier 0, also die Abstandsauswahl.
seedChoice	ArqRefinerTopDown	Hiermit wird die Auswahl des Seed-Beispiels beeinflusst. Die Standardeinstellung ist *random*, also wird das Seed-Beispiel zufällig gewählt. Es gibt noch *first* und *last*.

B Erläuterung zu den Datensets der URI

Tabelle B.1: Erläuterung zu den Datensets der URI

anneal.ORIG	Ein Datenset, dass das Abkühlverhalten von Metallen beschreibt
anneal	Wie anneal.ORIG, nur fehlende Daten anders dargestellt
audiology	Ein Datenset, indem Krankheitssymptome des Ohrs beschrieben werden
autos	Versicherungseinstufung von Autos anhand von Merkmalen der Autos
balance-scale	Ausgewogenheits-Problem von Psycholgischen Umfragen
breast-cancer	Einteilung von Brustkrebs in „wiederkehrend“ und „nicht wiederkehrend“
breast-w	Einteilung von Brustkrebs in „gutartig“ und „bösartig“
colic.ORIG	Datenset über Pferde mit Kolik
colic	Datenset über Pferde mit Kolik mit Diskretisierung einiger Daten
credit-a	Datenset über die Kreditbewilligung anhand von Attributen eines Menschen (aus Amerika)
credit-g	Datenset über die Kreditbewilligung anhand von Attributen eines Menschen (aus Deutschland)
diabetes	Einteilung in „keine Diabetes“ und „Diabetes“ anhand von Messdaten
glass	Glass anhand von verschieden Eigenschaften Klassifizieren
heart-c	Datenset zur Klassifikation einer Herzerkrankung
heart-h	Datenset zur Klassifikation einer Herzerkrankung
heart-statlog	Datenset zur Vorhersage des Vorhandenseins einer Herzkrankheiten
hepatitis	Leberentzündungs-Klassifikation in „überlebt“ und „stirbt“
hypothyroid	Schilddrüsenerkrankung anhand von Krankendaten erkennen
ionosphere	Radardaten der Ionosphere und Erkennung von Objekten
iris	Datenset zur Schwertlilien-Einteilung
kr-vs-kp	Schach-Endspiel : König und Turm gegen König und Bauer
labor	Vertragsverhandlungen und das Lernen eines guten Abschlußes
lymph	Erkennung von Lymphsystem-Erkrankungen
mushroom	Klassifikation von Pilzen in „essbar“ und „giftig“
primary-tumor	Anhand von Krankendaten einen Tumor in einem Körperteil vorhersagen
segment	Datenset zur Einteilung eines Bildes in mehrer Teile (Himmel,etc.)
sick	Datenset zur Erkennung von Schilddrüsen-Erkrankungen
sonar	Klassifikation von Sonarsignalen
soybean	Datenbank zur Erkennung von Krankheiten einer Sojabohne
splice	Erkennung von Problemen bei der Auftrennung von DNA-Strängen
vehicle	Klassifizieren einer gegebenen Silhouette als ein KFZ-Typ
vote	Datensammlung über Eigenschaften von Menschen und ihre Wahl zur Amerikanischen Kongresswahl
vowel	Hier sollen Vokale von verschiedenen Sprechern erkannt werden
zoo	Klassifikation von Tieren anhand von Merkmalen

C Detail Ergebnistabellen

Tabelle C.1: Vergleich AQR und CN2

Problem	**AQR**	**CN2**	**diff**
anneal	82,29	94,88	-12,59
anneal.ORIG	99,11	99,78	-0,67
audilogy	67,26	67,26	0,0
autos	77,56	75,12	2,44
balance-scale	34,24	83,84	-49,60
breast-cancer	57,69	68,53	-10,84
breast-w	94,13	95,57	-1,44
colic	77,45	78,26	8,42
colic.ORIG	78,53	70,11	-0,81
credit-a	81,30	82,46	-1,16
credit-g	66,90	71,40	-4,50
diabetes	68,75	68,23	0,52
glass	67,76	57,01	10,75
heart-c	72,61	72,94	-0,33
heart-h	74,83	74,83	0,0
heart-statlog	69,63	73,70	-4,07
hepatitis	80,65	78,06	2,59
hypothyroid	96,77	97,27	-0,50
ionosphere	88,60	93,16	-4,56
iris	94,00	96,00	-2,00
kr-vs-kp	96,40	98,90	-2,50
labor	84,21	87,21	-3,51
lymph	81,08	75,68	5,40
mushroom	97,76	100,00	-2,24
primary-tumor	18,29	39,23	-20,94
segment	92,27	86,93	5,54
sick	94,49	97,11	-2,62
sonar	76,44	64,42	12,02
soybean	84,77	89,60	-4,83
splice	88,68	52,73	35,95
vehicle	62,17	60,87	1,30
vote	92,87	94,94	-2,07
vowel	74,55	54,75	19,80
zoo	94,06	86,14	7,92

Tabelle C.2: AQR vs. AQR+W.C.

Problem	**AQR**	**AQR+W.C.**
anneal.ORIG	82,29	78,29
anneal	99,11	98,55
audiology	67,26	69,47
autos	77,56	71,22
balance-scale	34,24	31,20
breast-cancer	57,69	45,80
breast-w	94,13	93,28
colic.ORIG	78,53	73,10
colic	77,45	78,80
credit-a	81,30	68,12
credit-g	66,90	58,70
diabetes	68,75	58,72
glass	67,76	64,49
heart-c	72,61	71,95
heart-h	74,83	69,39
heart-statlog	69,63	70,74
hepatitis	80,65	80,65
hypothyroid	96,77	95,97
ionosphere	88,60	86,32
iris	94,00	96,67
kr-vs-kp	96,40	85,20
labor	84,21	89,47
lymph	81,08	79,05
mushroom	97,76	95,25
primary-tumor	18,29	13,57
segment	92,47	85,19
sick	94,49	93,85
sonar	76,44	75,00
soybean	84,77	79,21
splice	88,68	84,92
vehicle	62,17	60,17
vote	92,87	94,02
vowel	74,55	72,32
zoo	94,06	70,34

Tabelle C.3: Vergleich von AQR+W.C. und CN2+W.C.

Problem	**AQR+W.C.**	**CN2+W.C.**	**diff**
anneal	98,55	91,54	5,24
anneal.ORIG	78,29	73,05	7,01
audilogy	69,47	78,76	-9,29
autos	71,22	74,63	-3,41
balance-scale	31,20	83,84	-52,64
breast-cancer	45,80	70,28	-24,48
breast-w	93,28	95,28	-2,00
colic	78,80	80,71	-1,91
colic.ORIG	73,10	77,99	-4,89
credit-a	68,12	83,04	-14,92
credit-g	58,70	71,30	-12,6
diabetes	58,72	70,18	-11,46
glass	64,49	65,42	-0,93
heart-c	71,95	77,23	-5,28
heart-h	69,39	79,25	-9,86
heart-statlog	70,74	74,81	-4,07
hepatitis	80,65	76,13	4,52
hypothyroid	95,97	98,06	-2,09
ionosphere	86,32	91,74	-5,42
iris	96,67	96,67	0,00
kr-vs-kp	85,20	98,03	-12,83
labor	89,47	87,72	1,75
lymph	79,05	82,43	-3,38
mushroom	95,25	97,40	-2,15
primary-tumor	13,57	39,53	-25,96
segment	85,19	89,26	-4,07
sick	93,85	78,89	15,16
sonar	75,00	66,35	8,65
soybean	79,21	90,19	-10,98
splice	84,92	92,26	-7,34
vehicle	60,17	67,85	-7,68
vote	94,02	95,17	-1,15
vowel	72,32	61,01	11,31
zoo	70,34	90,10	-19,76

Tabelle C.4: AQR und CN2 jeweils ohne und mit W.C.

Problem	**AQR**	**CN2**	**diff**	**AQR+W.C.**	**CN2+W.C.**	**diff**
anneal	82,29	94,88	-12,59	98,55	91,54	5,24
anneal.ORIG	99,11	99,78	-0,67	78,29	73,05	7,01
audilogy	67,26	67,26	0,0	69,47	78,76	-9,29
autos	77,56	75,12	2,44	71,22	74,63	-3,41
balance-scale	**34,24**	83,84	-49,60	**31,20**	83,84	-52,64
breast-cancer	57,69	68,53	-10,84	45,80	70,28	-24,48
breast-w	94,13	95,57	-1,44	93,28	95,28	-2,00
colic	77,45	78,26	8,42	78,80	80,71	-1,91
colic.ORIG	78,53	70,11	-0,81	73,10	77,99	-4,89
credit-a	81,30	82,46	-1,16	68,12	83,04	-14,92
credit-g	66,90	71,40	-4,50	58,70	71,30	-12,6
diabetes	68,75	68,23	0,52	58,72	70,18	-11,46
glass	67,76	57,01	10,75	64,49	65,42	-0,93
heart-c	72,61	72,94	-0,33	71,95	77,23	-5,28
heart-h	74,83	74,83	0,0	69,39	79,25	-9,86
heart-statlog	69,63	73,70	-4,07	70,74	74,81	-4,07
hepatitis	80,65	78,06	2,59	80,65	76,13	4,52
hypothyroid	96,77	97,27	-0,50	95,97	98,06	-2,09
ionosphere	88,60	93,16	-4,56	86,32	91,74	-5,42
iris	94,00	96,00	-2,00	96,67	96,67	0,00
kr-vs-kp	96,40	98,90	-2,50	85,20	98,03	-12,83
labor	84,21	87,21	-3,51	89,47	87,72	1,75
lymph	81,08	75,68	5,40	79,05	82,43	-3,38
mushroom	97,76	100,00	-2,24	95,25	97,40	-2,15
primary-tumor	18,29	39,23	-20,94	13,57	39,53	-25,96
segment	92,27	86,93	5,54	85,19	89,26	-4,07
sick	94,49	97,11	-2,62	93,85	78,89	15,16
sonar	76,44	64,42	12,02	75,00	66,35	8,65
soybean	84,77	89,60	-4,83	79,21	90,19	-10,98
splice	88,68	52,73	35,95	84,92	92,26	-7,34
vehicle	62,17	60,87	1,30	60,17	67,85	-7,68
vote	92,87	94,94	-2,07	94,02	95,17	-1,15
vowel	74,55	54,75	19,80	94,02	95,17	-1,15
zoo	94,06	86,14	7,92	70,34	90,10	-19,76

Tabelle C.5: Ergebnisse gegen JRip

Problem	**AQR**	**CN2**	**AQR+W.C.**	**CN2+W.C.**	**JRip**	**Varianz**
anneal	82,29	94,88	98,55	91,54	98,33	11,27
anneal.ORIG	99,11	99,78	78,29	73,05	95,32	99,75
audilogy	67,26	67,26	69,47	78,76	72,89	24,59
autos	77,56	75,12	71,22	74,63	73,17	5,54
balance-scale	34,24	83,84	31,20	83,84	80,80	755,90
breast-cancer	57,69	68,53	45,80	70,28	70,98	117,68
breast-w	94,13	95,57	93,28	95,28	95,42	0,99
colic	77,45	78,26	78,80	80,71	82,61	7,35
colic.ORIG	78,53	70,11	73,10	77,99	84,24	24,02
credit-a	81,30	82,46	68,12	83,04	85,90	47,91
credit-g	66,90	71,40	58,70	71,30	71,70	30,96
diabetes	68,75	68,23	58,72	70,18	76,04	38,85
glass	67,76	57,01	64,49	65,42	68,69	21,24
heart-c	72,61	72,94	71,95	77,23	81,52	16,61
heart-h	74,83	74,83	69,39	79,25	78,91	15,98
heart-statlog	69,63	73,70	70,74	74,81	78,89	13,35
hepatitis	80,65	78,06	80,65	76,13	78,06	3,76
hypothyroid	96,77	97,27	95,97	98,06	99,34	1,66
ionosphere	88,60	93,16	86,32	91,74	89,74	7,14
iris	94,00	96,00	96,67	96,67	94,67	1,47
kr-vs-kp	96,40	98,90	85,20	98,03	99,19	34,62
labor	84,21	87,21	89,47	87,72	77,19	24,01
lymph	81,08	75,68	79,05	82,43	77,70	7,16
mushroom	97,76	100,00	95,25	97,40	100,00	3,99
primary-tumor	18,29	38,23	13,57	39,53	39,23	167,07
segment	92,27	86,93	85,19	89,26	95,15	16,38
sick	94,49	97,11	93,85	78,89	98,22	62,62
sonar	76,44	64,42	75,00	66,35	73,08	28,71
soybean	84,77	89,60	79,21	90,19	92,53	28,21
splice	88,68	52,73	84,92	92,26	94,45	292,08
vehicle	62,17	60,87	60,17	67,85	68,56	15,85
vote	92,87	94,94	94,02	95,17	95,40	1,08
vowel	74,55	54,75	94,02	95,17	69,09	68,37
zoo	94,06	86,14	70,34	90,10	87,13	81,83

Tabelle C.6: Anzahl an Regeln und Bedingungen Teil1

	CN2+Seed		Bexa		CN2		AQ	
Problem	Regeln	Bedingungen	Regeln	Bed.	Regeln	Bed.	Regeln	Bed.
anneal	11	18	18	43	12	20	10	17
anneal.ORIG	27	87	34	110	34	118	20	63
audiology	54	142	70	174	79	165	52	144
autos	33	75	32	62	44	81	36	85
balance-scale	32	104	47	152	51	166	81	576
breast-cancer	9	29	3	13	9	27	35	109
breast-w	21	50	20	45	21	49	11	30
colic	11	76	31	92	32	82	15	49
colic.ORIG	40	32	28	70	80	89	29	69
credit-a	37	76	45	128	51	133	29	103
credit-g	48	122	69	278	90	287	80	304
diabetes	83	284	71	220	83	255	47	233
glass	30	76	34	77	34	77	32	177
heart-c	13	43	20	52	22	58	17	63
heart-h	14	47	23	63	22	57	18	66
heart-statlog	17	53	20	51	19	48	14	55
hepatitis	6	15	13	31	10	23	8	27
hypothyroid	50	152	60	185	54	162	28	138
ionosphere	15	28	16	22	15	22	12	28
iris	8	11	6	11	6	11	3	3
kr-vs-kp	27	96	37	132	32	108	9	25
labor	4	5	4	6	4	5	5	8
lymph	10	28	8	23	11	26	8	22
mushroom	8	10	7	18	9	11	6	8
primary-tumor	48	220	52	255	63	274	23	97
segment	210	553	225	563	211	535	100	389
sick	54	177	47	152	47	146	24	106
sonar	17	33	28	44	29	45	11	36
soybean	59	194	61	298	68	217	63	214
splice	99	462	28	377	1519	1527	71	236
vehicle	112	323	172	445	187	495	131	501
vote	7	24	12	44	12	44	3	8
vowel	310	713	295	688	278	635	158	607
zoo	11	20	9	19	26	28	9	24

Tabelle C.7: Anzahl an Regeln und Bedingungen Teil2

	AQ+W.C.		JRip		Bexa+W.C.		CN2+W.C.		CN2+S+W.C.	
Problem	Regeln	Bedingungen	Regeln	Bed.	Regeln	Bed.	Regeln	Bed.	Regeln	Bed.
anneal	10	20	7	8	50	129	57	130	38	113
anneal.ORIG	9	16	14	37	60	192	82	229	53	155
audiology	69	252	15	27	105	293	111	284	85	231
autos	51	148	13	25	45	106	63	139	50	126
balance-scale	82	550	12	39	100	327	97	317	57	176
breast-cancer	30	98	3	4	9	40	23	72	1	4
breast-w	17	42	6	9	40	92	40	92	41	93
colic	8	24	4	6	41	145	59	173	24	86
colic.ORIG	41	129	5	9	34	93	95	143	31	72
credit-a	44	166	4	7	97	328	115	355	60	220
credit-g	192	873	3	5	160	667	130	455	97	361
diabetes	88	441	4	9	139	493	142	474	138	518
glass	36	147	8	18	43	116	43	116	35	93
heart-c	27	108	4	6	41	117	38	107	32	104
heart-h	38	147	3	4	53	172	48	144	25	91
heart-statlog	22	85	5	8	33	95	33	95	30	100
hepatitis	11	42	4	5	16	40	22	61	18	53
hypothyroid	39	208	5	11	50	157	48	151	57	218
ionosphere	21	53	3	2	20	33	20	33	31	55
iris	3	4	4	3	10	19	10	19	18	26
kr-vs-kp	46	134	16	44	113	395	110	363	87	344
labor	4	6	4	4	6	11	8	10	6	9
lymph	12	36	6	8	19	57	22	53	20	51
mushroom	13	27	9	12	30	66	72	88	58	107
primary-tumor	30	128	7	18	83	383	104	452	58	269
segment	151	682	21	53	244	727	250	759	215	597
sick	47	253	4	10	70	244	77	267	80	302
sonar	11	52	4	9	28	45	28	45	23	56
soybean	102	366	28	52	124	561	164	419	145	405
splice	86	386	14	55	58	813	1681	2409	231	1103
vehicle	174	765	17	43	240	747	255	802	235	777
vote	3	4	4	6	27	92	27	92	25	81
vowel	229	1028	51	145	315	870	323	872	409	1184
zoo	10	30	6	6	11	24	32	45	24	40

Tabelle C.8: Algorithmen ohne und mit Weighted Covering

	CN2+Seed.		Bexa		CN2		AQ	
	ohne	mit W.C.	ohne	mit W.C.	ohne	mit W.C.	ohne	mit W.C.
anneal	99,33	97,55	98,11	92,87	99,78	91,54	99,11	98,55
anneal.ORIG	95,43	81,07	94,21	91,98	94,88	73,05	82,29	78,29
audiology	71,24	75,66	65,49	75,22	67,26	78,76	67,26	69,47
autos	74,63	76,59	74,63	75,12	75,12	74,63	77,56	71,22
balance-scale	83,20	78,40	84,00	84,32	83,84	83,84	34,24	31,20
breast-cancer	70,63	70,63	74,13	73,08	68,53	70,28	57,69	45,80
breast-w	94,85	93,13	95,57	95,28	95,57	95,28	94,13	93,28
colic	77,17	77,99	76,63	79,89	78,26	80,71	77,45	78,80
colic.ORIG	74,18	76,90	80,43	81,52	70,11	77,99	78,53	73,10
credit-a	82,32	83,19	81,45	83,62	82,46	83,04	81,30	68,12
credit-g	71,50	72,50	67,20	65,70	71,40	71,30	66,90	58,70
diabetes	72,40	70,96	69,14	69,14	68,23	70,18	68,75	58,72
glass	61,22	50,93	59,81	67,76	57,01	65,42	67,76	64,49
heart-c	76,24	79,54	70,63	76,57	72,94	77,23	72,61	71,95
heart-h	80,61	77,89	75,51	76,53	74,83	79,25	74,83	69,39
heart-statlog	81,11	78,89	77,41	75,19	73,70	74,81	69,63	70,74
hepatitis	76,77	76,13	78,71	77,42	78,06	76,13	80,65	80,65
hypothyroid	97,11	96,24	97,19	98,22	97,27	98,06	96,77	95,97
ionosphere	92,31	89,74	93,45	91,74	93,16	91,74	88,60	86,32
iris	95,33	94,00	96,00	96,67	96,00	96,67	94,00	96,67
kr-vs-kp	99,22	95,71	99,12	98,09	98,90	98,03	96,40	85,20
labor	84,21	87,72	84,21	91,23	87,72	87,72	84,21	89,47
lymph	80,41	80,41	82,43	87,84	75,68	82,43	81,08	79,05
mushroom	100,00	95,91	100,00	100,00	100,00	97,40	97,76	95,25
primary-tumor	39,23	36,87	38,35	39,23	39,23	39,53	18,29	13,57
segment	87,84	86,80	87,27	90,04	86,93	89,26	92,47	85,19
sick	97,64	91,25	97,32	96,74	97,11	78,69	94,49	93,85
sonar	71,63	78,37	61,06	65,38	64,42	66,35	76,44	75,00
soybean	89,31	89,46	87,85	88,58	89,60	90,19	84,77	79,21
splice	88,15	89,91	89,12	91,50	52,73	92,26	88,68	84,92
vehicle	59,69	65,96	59,93	66,78	60,87	67,85	62,17	60,17
vote	93,10	91,49	95,40	95,17	94,94	95,17	92,87	94,02
vowel	50,51	61,11	48,69	59,19	54,75	61,01	74,55	72,32
zoo	88,12	89,11	86,14	87,13	86,14	90,10	94,06	70,34
Besser*	18	14	11	21	10	22	27	6

Tabelle C.9: AQ mit verschiedenen Parametern

	AQ	AQ Max	Diff	AQ Beamw. 30	Diff
anneal	99,11	99,33	0,22	98,66	-0,45
anneal.ORIG	82,29	91,76	9,47	83,63	1,34
audiology	67,26	74,34	7,08	70,80	3,54
autos	77,56	80,49	2,93	75,61	-1,95
balance-scale	34,24	63,20	28,96	35,36	1,12
breast-cancer	57,69	62,24	4,55	53,85	-3,84
breast-w	94,13	95,71	1,58	94,13	0,00
colic	77,45	82,88	5,43	82,34	4,89
colic.ORIG	78,53	81,25	2,72	80,43	1,90
credit-a	81,30	83,33	2,03	78,41	-2,89
credit-g	66,90	68,00	1,10	65,00	-1,90
diabetes	68,75	70,31	1,56	69,53	0,78
glass	67,76	68,69	0,93	64,02	-3,74
heart-c	72,61	76,57	3,96	73,27	0,66
heart-h	74,83	75,85	1,02	75,85	1,02
heart-statlog	69,63	77,04	7,41	74,81	5,18
hepatitis	80,65	85,16	4,51	85,16	4,51
hypothyroid	96,77	97,93	1,16	97,72	0,95
ionosphere	88,60	90,60	2,00	90,03	1,43
iris	94,00	97,33	3,33	96,00	2,00
kr-vs-kp	96,40	98,12	1,72	91,83	-4,57
labor	84,21	91,23	7,02	89,47	5,26
lymph	81,08	83,78	2,70	76,35	-4,73
mushroom	97,76	98,34	0,58	96,36	-1,40
primary-tumor	18,29	24,78	6,49	18,88	0,59
segment	92,47	94,46	1,99	93,25	0,78
sick	94,49	97,69	3,20	97,56	3,07
sonar	76,44	80,77	4,33	76,92	0,48
soybean	84,77	90,63	5,86	86,53	1,76
splice	88,68	90,97	2,29	90,75	2,07
vehicle	62,17	67,85	5,68	67,73	5,56
vote	92,87	95,63	2,76	95,63	2,76
vowel	74,55	81,01	6,46	77,58	3,03
zoo	94,06	94,06	0,00	93,07	-0,99

Tabelle C.10: CN2 mit verschiedenen Parametern

	beamwidth 1	beamwidth 2	beamwidth 4	beamwidth 5	beamwidth 10
anneal.ORIG	95,10	94,88	95,10	94,65	95,32
anneal	99,44	99,67	99,55	99,44	99,44
audiology	62,83	64,60	65,93	69,47	69,47
autos	77,56	77,07	77,07	77,56	80,49
balance-scale	84,16	83,36	84,64	83,20	83,20
breast-cancer	72,03	72,73	69,58	71,68	70,28
breast-w	94,99	95,57	94,42	95,28	95,99
colic.ORIG	71,20	72,55	70,11	72,83	74,46
colic	76,90	77,99	75,82	77,45	79,35
credit-a	81,01	78,99	80,29	80,43	80,29
credit-g	72,70	71,90	70,80	70,40	71,20
diabetes	69,53	70,18	69,92	70,70	67,19
glass	62,62	61,68	62,62	63,55	61,68
heart-c	74,92	73,60	76,24	74,59	71,95
heart-h	75,51	74,15	75,17	76,19	73,47
heart-statlog	75,93	74,07	75,93	73,70	74,07
hepatitis	78,06	82,58	76,77	78,71	79,35
hypothyroid	97,22	97,32	96,66	96,87	97,03
ionosphere	94,02	91,74	94,02	93,73	92,31
iris	96,00	95,33	96,00	95,33	94,67
kr-vs-kp	99,06	98,97	99,06	99,00	99,16
labor	89,47	91,23	85,96	87,72	87,72
lymph	77,70	77,70	80,41	75,68	77,03
mushroom	100,00	100,00	100,00	100,00	100,00
primary-tumor	40,41	36,87	36,58	38,05	36,28
segment	85,76	85,54	87,32	86,58	87,27
sick	97,61	97,19	97,03	96,98	96,95
sonar	64,90	67,31	64,90	63,94	68,27
soybean	90,92	89,75	89,17	89,75	88,87
splice	52,60	52,57	52,85	52,92	53,20
vehicle	61,23	60,87	62,88	60,28	60,28
vote	96,09	95,86	95,63	95,40	95,40
vowel	51,41	52,73	53,23	52,63	56,26
zoo	78,22	78,22	86,14	84,16	84,16

Tabelle C.11: AQ mit W.C. und verschiedenen Parametern

	AQ+W.C.	AQ+W.C. Max	Diff	AQ+W.C. Beam 40	Diff
anneal	98,55	98,89	0,34	98,44	8,57
anneal.ORIG	78,29	92,09	13,80	86,86	-0,11
audiology	69,47	80,09	10,62	70,35	0,88
autos	71,22	80,49	9,27	75,12	3,90
balance-scale	31,20	46,72	15,52	34,56	3,36
breast-cancer	45,80	54,90	9,10	44,06	-1,74
breast-w	93,28	95,28	2,00	92,27	-1,01
colic	78,80	82,34	3,54	74,73	-2,18
colic.ORIG	73,10	76,36	3,26	70,92	-4,07
credit-a	68,12	77,83	9,71	72,75	4,63
credit-g	58,70	73,62	14,92	72,75	14,05
diabetes	58,72	65,10	6,38	65,10	6,38
glass	64,49	69,16	4,67	64,95	0,46
heart-c	71,95	76,24	4,29	73,27	1,32
heart-h	69,39	74,15	4,76	70,07	0,68
heart-statlog	70,74	73,33	2,59	68,89	-1,85
hepatitis	80,65	80,65	0,00	78,71	-1,94
hypothyroid	95,97	97,24	1,27	97,19	1,22
ionosphere	86,32	89,74	3,42	86,61	0,29
iris	96,67	96,67	0,00	96,00	-0,67
kr-vs-kp	85,20	90,80	5,60	86,17	0,97
labor	89,47	92,98	3,51	92,98	3,51
lymph	79,05	79,73	0,68	76,35	-2,70
mushroom	95,25	96,23	0,98	93,13	-2,12
primary-tumor	13,57	14,45	0,88	14,45	0,88
segment	85,19	87,92	2,73	86,80	1,61
sick	93,85	96,92	3,07	95,68	1,83
sonar	75,00	80,29	5,29	77,40	2,40
soybean	79,21	89,75	10,54	81,99	2,78
splice	84,92	87,55	2,63	87,46	2,54
vehicle	60,17	65,72	5,55	65,72	5,55
vote	94,02	94,02	0,00	86,67	-7,35
vowel	72,32	81,62	9,30	77,68	5,36
zoo	94,06	95,05	0,99	94,06	0,00

Tabelle C.12: AQ mit der Heuristik Correlation

	AQ	AQ+Corr	AQ+W.C.	AQ+W.C.+Corr
anneal	99,11	98,44	98,55	93,32
anneal.ORIG	82,29	85,08	78,29	73,72
audiology	67,26	69,47	69,47	73,45
autos	77,56	78,05	71,22	78,05
balance-scale	34,24	35,36	31,20	38,24
breast-cancer	57,69	52,80	45,80	55,94
breast-w	94,13	93,71	93,28	91,99
colic	77,45	80,43	78,80	76,36
colic.ORIG	78,53	70,92	73,10	72,83
credit-a	81,30	77,97	68,12	73,33
credit-g	66,90	61,00	58,70	57,40
diabetes	68,75	65,49	58,72	51,30
glass	67,76	63,55	64,49	64,02
heart-c	72,61	75,25	71,95	70,96
heart-h	74,83	73,47	69,39	70,07
heart-statlog	69,63	74,07	70,74	77,04
hepatitis	80,65	76,13	80,65	76,13
hypothyroid	96,77	97,11	95,97	94,88
ionosphere	88,60	85,19	86,32	87,18
iris	94,00	94,00	96,67	94,00
kr-vs-kp	96,40	92,58	85,20	83,79
labor	84,21	89,47	89,47	85,96
lymph	81,08	78,38	79,05	71,62
mushroom	97,76	96,97	95,25	93,39
primary-tumor	18,29	13,57	13,57	4,72
segment	92,47	92,47	85,19	89,35
sick	94,49	90,96	93,85	90,72
sonar	76,44	78,37	75,00	77,88
soybean	84,77	85,51	79,21	82,58
splice	88,68	87,65	84,92	76,90
vehicle	62,17	62,41	60,17	56,50
vote	92,87	94,48	94,02	94,48
vowel	74,55	71,52	72,32	65,45
zoo	94,06	90,10	70,34	94,06

Tabelle C.13: CN2 mit W.C. und verschiedenen Parametern

	beamwidth 1	beamwidth 2	beamwidth 4	beamwidth 5	beamwidth 10	weight 0.25	weight 0.33	weight 0.85
anneal.ORIG	74,16	73,61	72,94	72,72	73,61	78,17	77,17	72,61
anneal	91,43	91,54	91,98	91,31	92,32	95,66	93,32	88,2
audiology	76,11	76,55	75,66	75,66	78,76	79,2	77,43	69,03
autos	69,76	67,8	69,76	67,8	66,83	78,05	78,05	59,51
balance-scale	83,04	83,52	83,84	84,16	83,2	84,64	84,64	81,6
breast-cancer	71,33	72,73	70,98	70,98	69,23	68,88	70,28	71,33
breast-w	96,14	95,57	95,28	95,57	95,14	95,28	95,71	95,71
colic.ORIG	78,26	77,45	76,36	75,27	75,82	77,45	74,46	77,99
colic	80,43	79,08	81,25	82,61	79,35	81,25	79,62	79,62
credit-a	83,77	82,61	83,04	83,04	83,62	83,48	82,75	81,88
credit-g	73,6	71,6	70,5	70,5	69,1	72	72,5	68
diabetes	69,14	67,58	67,45	68,75	68,1	69,92	67,58	63,93
glass	64,02	65,89	64,95	65,89	66,36	64,02	62,62	64,95
heart-c	76,24	74,92	76,9	75,58	76,9	77,56	74,92	74,26
heart-h	79,25	78,23	78,91	80,27	78,57	79,25	78,23	77,21
heart-statlog	74,81	75,56	74,81	77,04	75,19	77,41	77,41	74,44
hepatitis	78,06	78,71	78,06	75,48	75,48	79,35	80	73,55
hypothyroid	97,75	98,25	98,22	98,22	98,49	97,96	98,12	93,32
ionosphere	92,31	92,02	92,02	91,45	90,03	91,74	91,17	91,45
iris	96,67	96,67	96,67	96,67	96,67	96,67	96,67	96
kr-vs-kp	98,62	98,53	97,81	98,09	98,12	99,19	98,87	96,87
labor	91,23	87,72	87,72	87,72	87,72	89,47	87,72	91,23
lymph	81,08	80,41	81,76	79,05	82,43	80,41	81,76	78,38
mushroom	96,7	97,35	97,18	97,17	97,62	98,72	98,17	96,57
primary-tumor	38,35	37,76	37,17	39,23	37,76	38,35	37,46	35,4
segment	90,22	91,95	91,9	92,86	92,08	90,48	91,17	92,16
sick	79,83	78,34	78,37	78,61	78,47	79,35	80,06	77,81
sonar	65,38	65,38	67,31	67,31	73,08	65,87	71,63	68,75
soybean	90,78	91,51	90,92	91,65	90,19	91,51	92,24	88,58
splice	91,82	92,45	92,48	92,1	92,57	93,57	94,04	88,78
vehicle	67,85	67,97	69,5	70,21	70,45	71,28	69,74	65,96
vote	95,4	94,94	95,17	94,94	94,94	94,94	95,86	94,48
vowel	57,17	60,4	62,93	65,35	67,07	61,41	63,33	64,24
zoo	90,1	90,1	87,13	88,12	87,13	90,1	88,12	87,13

Tabelle C.14: Win-Loose-Vergleich des W.C.

Win/Loose	CN2+Seed+W.C.	CN2+Seed+W.C.+Redundanz	Bexa+W.C.	Bexa+W.C.+Redundanz	AQ+W.C.	AQ+W.C.+Redundanz	CN2+W.C.	CN2+W.C.+Redundanz	JRIP	Summe
A	-/-	10/21	12/22	10/24	24/10	23/9	12/20	9/22	7/25	107/153
B	21/10	-/-	14/20	11/21	24/10	20/14	16/18	14/19	13/21	133/132
C	22/12	20/14	-/-	11/17	29/4	28/6	17/13	15/13	8/23	150/102
D	24/10	21/11	17/11	-/-	29/4	29/5	22/9	20/12	10/22	172/84
E	10/24	10/24	4/29	4/29	-/-	14/19	7/26	7/26	7/27	63/204
F	9/23	14/20	6/28	5/29	19/14	-/-	7/25	8/24	6/27	74/190
G	19/12	17/16	13/17	9/22	25/7	24/7	-/-	12/14	10/24	133/119
H	22/9	19/14	13/15	12/20	26/7	24/8	14/12	-/-	11/23	141/108
I	25/7	21/13	23/8	22/10	27/7	27/6	24/10	23/11	-/-	192/72
Summe vs.	152/107	132/133	102/150	84/172	204/63	190/74	119/133	108/141	72/192	-/-

Tabelle C.15: Win-Loose Aufstellung mit geänderter Klassifikation für W.C.

Win/Loose	CN2+Seed	Bexa	CN2	AQ	AQ+W.C.	JRip	Bexa+W.C.	CN2+W.C.	CN2+S+W.C.	Summe
CN2+Seed	-/-	19/12	17/15	21/12	24/10	12/20	13/19	12/22	16/17	134/127
Bexa	12/19	-/-	15/15	21/12	23/10	9/23	14/18	14/20	16/17	124/134
CN2	15/17	15/15	-/-	20/12	22/11	7/24	8/23	9/24	15/18	111/144
AQ	12/21	12/21	12/20	-/-	23/10	8/26	8/25	11/22	14/19	100/164
AQ+W.C.	10/24	10/23	11/22	10/23	-/-	6/27	5/29	8/24	14/20	74/192
JRip	20/12	23/9	24/7	26/8	27/6	-/-	22/10	23/11	21/13	186/76
Bexa+W.C.	19/13	18/14	23/8	25/8	29/5	10/22	-/-	20/12	21/11	165/93
CN2+W.C.	22/12	20/14	24/9	22/11	24/8	11/23	12/20	-/-	19/14	154/111
CN2+S+W.C.	17/16	17/16	18/15	19/14	20/14	13/21	11/21	14/19	-/-	129/136
Summe vs.	127/134	134/124	144/111	164/100	192/74	76/186	93/165	111/154	136/129	-/-

Danksagung

Danken möchte ich Prof. Dr. Johannes Fürnkranz für die gute Betreuung und seine guten Tipps während meiner Arbeit.

Auch danken möchte ich Dr. Gunter Grieser für seine fundierten LaTeXKenntnisse die er mir zur Verfügung gestellt hat.

Meiner Freundin Martina danke ich für die Unterstützung das ganze Studium hinweg.

Ich danke auch meinen Komilitonen und Freunden, die in den Genuss kamen diese Arbeit Korrektur zu lesen und trotzdem noch meine Freunde sind.

Vor allem aber möchte ich meinen Eltern und Großeltern danken, die mir dieses Studium überhaupt erst ermöglicht haben.