Implementierung und Evaluierung eines Greedy Association Rule Learners

Implementation and Evaluation of a Greedy Association Rule Learner

Bachelor-Thesis von Johannes Dorn

März 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Implementierung und Evaluierung eines Greedy Association Rule Learners
Implementation and Evaluation of a Greedy Association Rule Learner

Vorgelegte Bachelor-Thesis von Johannes Dorn

Gutachten: Johannes Fürnkranz

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

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

Darmstadt, den 1. März 2010

(J. Dorn)

Zusammenfassung

In dieser Bachelorarbeit wird ein Algorithmus eines Assoziationslerners vorgestellt, der typische Techniken von Klassifizierern einsetzt. Die Hill-Climbing Strategie wird in einigen Separate-and-Conquer Klassifizierern verwendet, um nicht alle möglichen Regeln zu evaluieren, und soll diese Aufgabe im vorgestellten Algorithmus für einen Assoziationslerner übernehmen. Der Algorithmus kann zum Bewerten der Qualität der gefundenen Regeln verschiedene Evaluationsmetriken einsetzen, welche im einzelnen hier vorgestellt und deren Auswirkungen auf das Ergebnis diskutiert werden. Um die Performanz zu steigern, wird eine parallelisierte Variante des Algorithmus vorgestellt, welche auf Multiprozessorsystemen zu einer erheblichen Leistungssteigerung führen kann. Um die Praxistauglichkeit des Algorithmus zu überprüfen, wird er mit einem klassischen Assoziationslerner verglichen. Dazu werden die gefundenen Regeln gegenübergestellt und überprüft, welche Unterschiede zwischen dem vorgestellten und dem klassischen Assoziationslerner bestehen. Zuletzt wird ein Vergleich auf Basis der Performanz durchgeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

1 Einleitung

Dieses Kapitel soll die Motivation hinter der Entwicklung eines Greedy Association Rule Learners erläutern.

1.1 Regeln

Im Bereich des maschinellen Lernens gibt es zwei Lerner-Typen: Klassifizierer sowie Assoziationslerner. Beide finden Regeln der Form $X \Rightarrow Y$, wobei X und Y eine Menge von Attributen bzw. deren Werte aus einem Beispieldatensatz sind. X und Y haben dabei keine gemeinsamen Attribute. Im Folgenden wird X als die Menge der Prämissen und Y als die Menge der Konklusionen bezeichnet. Klassifizierer finden immer genau ein Element in Y, Assoziationslerner können mehrere Attribute in der Konklusion einer Regel finden.

1.2 Klassifizierer

Klassifizierer versuchen Regeln zu finden, die ein festes Attribut eines Datensatzes vorhersagen. Es gibt somit nur ein Attribut in der Konklusion, welches aber alle Werte des Beispieldatensatzes annehmen kann. Tabellen 1 und 2 zeigen einen Beispieldatensatz und zwei darin gefundene Klassifikationsregeln.

Temperature	**Outlook**	**Humidity**	**Windy**	**Play Golf?**
mild	overcast	normal	false	yes
mild	rain	high	true	no
hot	overcast	high	false	yes
cool	rain	normal	false	no

Tabelle 1: Beispieldatensatz für einen Klassifizierer

{Temperature = mild, Windy = false}	⇒	{Play Golf = yes}
{Outlook = rain}	⇒	{Play Golf = no}

Tabelle 2: Beispiel zweier Klassifikationsregeln

Eine populäre Form von Klassifizierern arbeitet mit separate-and-conquer Algorithmen [3]. Diese suchen nach einer Regel, die Teile des Beispieldatensatzes erklärt, trennen diese Beispiele vom Rest und beginnen mit den übrigen von vorne. Diese Technik stellt sicher, dass jedes Beispiel von mindestens einer Regel erklärt wird.
Separate-and-conquer Algorithmen können die hill-climbing Suche verwenden, welche beginnend mit einer leeren Regel diese immer weiter verfeinert, bis sich die Qualität der Regel nicht mehr verbessern lässt.

1.3 Assoziationslerner

Von Assoziationslernern gefundene Regeln haben eine variable Anzahl an Prämissen und Konklusionen. Eine häufige Anwendung stellt die Analyse von Einkaufslisten dar. Assoziationslerner finden Zusammenhänge zwischen gekauften Produkten, also welche Produkte in Kombination erworben werden [4]. Die Motivation dahinter kann sein, die Anordnung der Waren in einem Supermarkt zu optimieren. Tabellen 3 und 4 zeigen einen Beispieldatensatz sowie zwei darin gefundene Assoziationsregeln.

Einkauf 1	Brot	Windeln	Bier	Flips	
Einkauf 2	Chips	Cola	Flips		
Einkauf 3	Windeln	Cola	Flips		
Einkauf 4	Chips	Wasser	Cola	Bier	Flips

Tabelle 3: Beispieldatensatz für einen Assoziationslerner

{beer, wine}	⇒	{chips}
{diapers}	⇒	{beer, chips}

Tabelle 4: Beispiel zweier Assoziationsregeln

Klassische Assozationslerner arbeiten in zwei Schritten. Zuerst werden frequent Itemsets gefunden. Itemsets sind Mengen von einzelnen Attributen in einem Beispieldatensatz. Sie sind frequent (häufig), wenn sie mindestens so oft Teil von Beispielen sind, wie vom Benutzer festgelegt. Im zweiten Schritt werden Regeln erstellt, die aus den Attributen der einzelnen frequent Itemsets bestehen.

Im Folgenden wird der hier vorgestellte Algorithmus mit Apriori [1] verglichen. Apriori benötigt zwei Eingaben vom Benutzer: Minimum Support und Minimum Confidence. Minimum Support bezeichnet die minimale Häufigkeit, mit der ein Itemset vorkommen muss um als frequent zu gelten und somit für eine Regel in Betracht gezogen zu werden. Confidence ist eine Evaluationsmetrik, die die Wahrscheinlichkei berechnet, mit der die Konklusion einer gefundenen Regel für die entsprechende Prämisse eintrifft. Der minimum Wert soll sicherstellen, dass nur „gute" Regeln gelernt werden. Alternativ zu Confidence können auch andere Metriken verwendet werden. Mehr dazu in Kapitel 2.4.

1.4 Ziele

Ziel dieser Bachelorarbeit ist die Implementation eines Assoziationslerners, der Techniken von separate-and-conquer Klassifizierern einsetzt. Dieser Lerner soll anschließend evaluiert werden. Zu klären ist dabei, wie hoch die Qualität der gelernten Regeln ist und ob die Laufzeit des Algorithmus praxistauglich ist.

1.5 Übersicht

Zuerst betrachten wir in Abschnitt 2 die Arbeitsweise des vorgestellten Algorithmus inklusive der Wahl von Beispielbewertung und einer Diskussion verschiedener Evaluationsmetriken.
Mit Optimierungsmöglichkeiten des Algorithmus beschäftigt sich Abschnitt 3 und soll Möglichkeiten aufzeigen, wie die Implementation des Algorithmus bezüglich der Laufzeit verbessert werden kann.
Zuletzt findet in Abschnitt 4 eine Evaluation des Algorithmus statt. Dazu wird sowohl die Qualität der Regeln als auch die Laufzeit des Algorithmus mit einer Apriori Implementation verglichen.

2 Greedy Association Rule Learner

In diesem Kapitel wird ein Algorithmus für einen Assoziationslerner vorgestellt, der separate-and-conquer Techniken von Klassifizierern benutzt. Der Algorithmus basiert auf einem generischen separate-and-conquer Lerner, der in [3] vorgestellt wurde.

Ein Brute-Force Ansatz zum Erstellen und Bewerten aller möglichen Regeln ist schon bei kleinen Datensätzen zu aufwändig, da die Anzahl der möglichen Regeln exponentiell ansteigt.

$$Num_{rules} = 3^d - 2^{d+1} + 1$$

Anzahl der möglichen Assoziationsregeln für d verschiedene Attribute in einem Beispieldatensatz.

Von der Summe aller Möglichkeiten, die Attribute in Prämisse und Konklusion anzuordnen (3^d), werden alle Regeln abgezogen, bei denen einzelne Attribute sowohl in der Prämisse als auch der Konklusion vorkommen ($-2^{d+1} + 1$). Die Formel gilt für Attribute, die nur zwei Werte annehmen können, was beispielsweise beim Lernen von Regeln in Einkaufsdaten der Fall ist.

Angenommen ein Supermarkt hätte nur die folgenden Waren im Angebot: $\{Bier, Chips, Windeln\}$, gäbe es insgesamt $3^3 - 2^{3+1} + 1 = 12$ mögliche Regeln, welche in Tabelle 5 gezeigt werden.

#	Prämisse	Konklusion
1	Bier	Chips
2	Bier	Windeln
3	Bier	Chips && Windeln
4	Chips	Bier
5	Chips	Windeln
6	Chips	Bier && Windeln
7	Windeln	Bier
8	Windeln	Chips
9	Windeln	Bier && Chips
10	Bier && Chips	Windeln
11	Bier && Windeln	Chips
12	Chips && Windeln	Bier

Tabelle 5: Mögliche Regeln für $\{Bier, Chips, Windeln\}$

Abbildung 1 verdeutlicht, wie drastisch die Anzahl der möglichen Regeln für zunehmende Attribute steigt.

Der vorgestellte Algorithmus, im folgenden GARL genannt, ist „greedy" (gierig), da die hill-climbing Strategie angewendet wird und somit nicht alle möglichen Regeln in Betracht gezogen werden.

2.1 Arbeitsweise

GARL erwartet eine Reihe von Parametern.

- **examples** - Ein Beispieldatensatz
- **coverageNumber** - Gibt an, wie oft jedes Beispiel mindestens abgedeckt werden muss.
- **minimumEvaluation** - Gibt an, wie hoch der Evaluationswert einer Regel mindestens sein muss, um zur Theorie hinzugefügt werden.
- **randomSeed** - Ein Seed Wert für den Pseudozufallsgenerator .
- **evaluator** - Die zu verwendene Evaluationsmetrik. Mehr dazu in Abschnitt 2.2.
- **maxNoImprovement** - Gibt an, nach wie vielen Verfeinerungsschritten ohne Verbesserung für das aktuelle Beispiel abgebrochen wird.
- **NoRedundancyCheck** - Optionales Flag, das das Finden von redundanten Regeln zulässt.

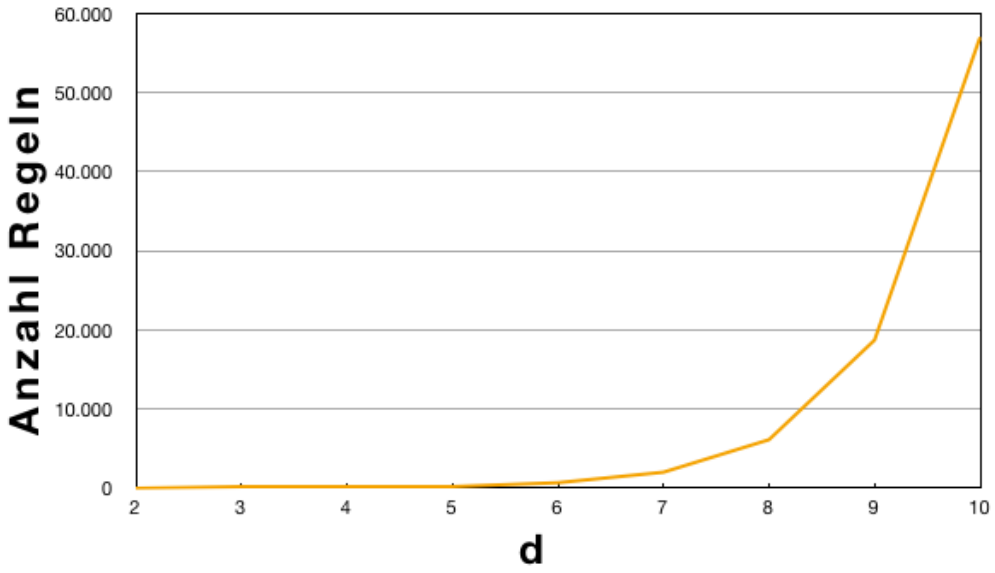

Abbildung 1: Anzahl der möglichen Assoziationsregeln für d Attribute

Algorithmus 1 Pseudocode für den Greedy Association Rule Learner

```
1: theory = φ
2: while (!allExamplesCovered) do
3:    example = chooseRandomExample(examples)
4:    rule = findBestRule(examples, example)
5:    markExamplesAsCovered(examples, rule)
6:    if (evaluation(rule) > minimumEvaluation) then
7:       theory = theory ∪ rule
8:    end if
9: end while
10: return theory
```

Algorithmus 1 zeigt die äußere Schleife von GARL.

Einer anfangs leeren Theorie werden Regeln hinzugefügt, bis alle Beispiele minimal abgedeckt wurden. Innerhalb der Schleife wird ein zufälliges Beispiel aus dem Datensatz ausgewählt und mit dessen Attributen die bestmögliche Regel mit Hilfe der hill-climbing Strategie gefunden. Alle Beispiele haben einen Zähler, der speichert, wie oft sie bereits abgedeckt wurden. Der Zähler jedes Beispiels, das von der gefundenen Regel abgedeckt wird, wird inkrementiert. Ein Beispiel gilt als abgedeckt, falls die Prämisse der Regel im Beispiel zu finden ist. Sollte der Evaluationswert der Regel größer sein als der verlangte minimumEvaluation Wert, wird die Regel zur Theorie hinzugefügt.

Zu beachten ist, dass Beispiele auch dann als abgedeckt markiert werden, falls der minimumEvaluation Wert nicht erreicht wurde. Damit wird sichergestellt, dass der Algorithmus auch terminiert, wenn es nicht genügend „gute" Regeln gibt, um die gewünschte minimale Abdeckung zu erreichen. Das hat allerdings auch zur Folge, dass die vom Algorithmus zurückgegebenen Regeln nicht immer ausreichen, um die Beispiele gemäß des coverageNumber Wertes minimal abzudecken.

2.2 Beispielwahl

Die Methode `chooseRandomExample(examples)` wählt mit Hilfe eines Pseudozufallsgenerator ein zufälliges Beispiel aus dem Datensatz. Dabei werden allerdings nur die Beispiele in Betracht gezogen, die am seltensten abgdeckt wurden.

Um reproduzierbare Ergebnisse zu erhalten, arbeitet der Generator mit einem, vom Benutzer übergebenen, Seed Wert.

2.3 Regel finden

Algorithmus 2 zeigt den Pseudocode für die Methode `findBestRule(examples, example)`.

Die Methode versucht die bestmögliche Regel für ein gegebenes Beispiel zu finden. Begonnen wird mit einer leeren Regel, also einer Regel ohne Prämisse oder Konklusion. Diese Regel wird nun nach und nach verfeinert, bis keine Verfeinerung mehr möglich ist. Zuerst werden in `refineRule(candidate, examples, example)` alle möglichen Verfeinerungen erstellt. Verfeinerungen sind alle Möglichkeiten, die ursprüngliche Regel um eine Prämisse oder Konklusion aus dem Beispiel zu erweitern, welche noch nicht Teil der Regel sind. Soll die leere Regel verfeinert werden, werden alle Kombinationsmöglichkeiten aus je einem Beispielattribut in Prämisse und Konklusion erzeugt.

Aus den Verfeinerungen werden alle bereits gelernten Regeln entfernt, um sicher zu stellen, dass der Algorithmus für coverageNumber Werte größer eins nicht immer wieder die selben Regeln findet. Die übrig gebliebenen werden mit einer Evaluationsmetrik bewertet.

`findBestRule(examples, example)` unterscheidet zwischen der besten Regel aus den aktuellen Verfeinerungen (currentBestRule) und der besten Regel aus dem kompletten Durchlauf (overallBestRule). Dadurch können auch Verbesserungen von Regeln gefunden werden, wenn in den vorherigen Verfeinerungsschritten nur niedriger bewertete Regeln gefunden wurden. Da eine Suche bis hin zu Regeln mit allen Attributen häufig keine Verbesserungen bietet, wird über einen vom Benutzer übergebenen Wert `maxNoImprovement` angegeben, wie viele Verfeinerungsschritte gemacht werden, nachdem keine neue beste Regel gefunden wurde.

Algorithmus 2 Pseudocode der Methode findBestRule(examples, example)

```
1:  currentBestRule = emptyRule()
2:  overallBestRule = currentBestRule
3:  evaluate(currentBestRule)
4:  rules = {currentBestRule}
5:  noImprovementCounter = 0;
6:  while rules ≠ φ
    && isImprovementPossible(overallBestRule)
    && noImprovementCounter < maxNoImprovement do
7:    newOverallBestRule = false
8:    candidates = selectCandiates(rules)
9:    rules = rules \ candidates
10:   for candidate ϵ candidates do
11:     refinements = refineRule(candidate, examples, example)
12:     refinements = refinements \ theory
13:     for refinement ϵ refinements do
14:       evaluate(refinement)
15:       if !isRefinementRedundant(refinement) then
16:         rules = rules ∪ refinement
17:         if (refinement > currentBestRule) then
18:           currentBestRule = refinement
19:           if (refinement > overallBestRule) then
20:             overallBestRule = refinement
21:             newOverallBestRule = true
22:           end if
23:         end if
24:       end if
25:     end for
26:   end for
27:   rules = filterRules(rules)
28:   currentBestRule = emptyRule()
29:   evaluate(curentBestRule)
30:   if !newOverallBestRule then
31:     noImprovementCounter = noImprovementCounter + 1
32:   end if
33: end while
34: return  overallBestRule
```

2.4 Regelbewertung

Die Bewertung von Regeln ist ein elementarer Bestandteil der Hill-Climbing Strategie. GARL kann mit verschiedensten Evaluationsmetriken arbeiten. Im praktischen Teil dieser Bachelorarbeit wurden mehrere implementiert. Die Berechnung der Evaluationsmetriken und deren Folgen für die gefundenen Regeln werden hier erläutert.

2.4.1 Kenngrößen

Die Berechnung von Evaluationsmetriken erfolgt aus einer Reihe von Kenngrößen.

- p - Die Anzahl der Beispiele mit zutreffender Prämisse und Konklusion. Wird auch als true positives bezeichnet.
- n - Die Anzahl der Beispiele mit zutreffender Prämisse aber nicht zutreffender Konklusion. Wird auch als false positives bezeichnet.
- P - Die Anzahl der Beispiele mit zutreffender Konklusion. Die Prämisse spielt für P keine Rolle.
- N - Die Anzahl der Beispiele mit nicht zutreffender Konklusion. Die Prämisse spielt für N keine Rolle.

2.4.2 Confidence

$$h_{conf} = \frac{p}{p+n}$$

Confidence, manchmal auch Precision genannt, berechnet den Anteil von positiven Beispielen unter allen abgedeckten Beispielen. Der Wert stellt somit dar, wie viel Vertrauen (Confidence) die Metrik hat, dass die Konklusion auf die Prämisse folgt.
Confidence ist eine der simpelsten Metriken und wird im Folgenden dazu verwendet werden, die Qualität der gefundenen Regeln in GARL und Apriori zu vergleichen. Ein Nachteil von Confidence ist, dass die Anzahl der abgedeckten Beispiele nur eine geringe Rolle spielt, insbesondere, wenn alle abgedeckten Beispiele positiv sind. So bekommt eine Regel mit nur einem positiven und keinem negativen Beispiel den gleichen Evaluationswert wie eine Regel mit 100 positiven, aber keinen negativen Beispielen.

$$h_{conf} = \frac{1}{1+0} \equiv \frac{100}{100+0} = 1$$

2.4.3 Laplace

$$h_{Lap} = \frac{p+1}{(p+1)+(n+1)} = \frac{p+1}{p+n+2}$$

Laplace ist Confidence sehr ähnlich, stellt aber sicher, dass der maximale Wert von 1 niemals erreicht werden kann. Somit werden 100-prozentig positive Regeln besser bewertet, wenn sie mehr Beispiele haben.

$$h_{Lap} = \frac{1+1}{1+0+2} = \frac{2}{3} < \frac{100+1}{100+0+2} = \frac{101}{102} \approx 0.99$$

2.4.4 Weighted Relative Accuracy

$$h_{WRA} = \frac{p+n}{P+N}\left(\frac{p}{p+n} - \frac{P}{P+N}\right)$$

WRA besteht aus zwei Komponenten: $\frac{p+n}{P+N}$, die Anzahl der abgedeckten Beispiele, sowie $(\frac{p}{p+n} - \frac{P}{P+N})$. Die zweite Komponente gibt an, wie viel präziser die Regel gegenüber der allgemeinen Regel ist, welche davon ausgeht, dass alle Beispiele die Konklusion erfüllen.

2.4.5 Lift

$$h_{Lift} = \frac{\frac{p}{p+n}}{\frac{P}{P+N}}$$

Lift, früher auch Interest gennant, berechnet, wie viel öfter Prämisse und Konklusion zusammen auftreten, als wenn sie statistisch unabhängig wären.

2.4.6 Correlation

$$h_{Corr} = \frac{p(N-n) - (P-p)n}{\sqrt{PN(p+n)(P-p+N-n)}}$$

Correlation berechnet einen Korrelations-Koeffizienten zwischen der Vorhersage der Regel und den tatsächlichen Werten in abgedeckten Beispielen.

2.5 Redundanzkriterium

Die Menge aller möglichen Assoziationsregeln enthält einen großen Teil an redundanten Regeln, die keinen Mehrwert gegenüber anderen Regeln bieten. In der Literatur existiert allerdings noch keine einheitliche Definition für redundante Regeln. Nach [10] sind redundante Regeln solche, die zusätzliche Attribute in Prämisse oder Konklusion (aber nicht beiden) bei gleichem Evaluationswert haben.

Für diese Arbeit wird die Definition aus [2] verwendet, welche besagt, dass nicht redundante Regeln eine minimale Anzahl an Elementen in der Prämisse und eine maximale Anzahl in der Konklusion haben.

Seien in den Verkaufsdaten eines Supermarktes die in Tabelle 6 dargestellten Regeln gefunden worden. Angenommen, alle Regeln wurden mit dem maximalen Confidence Wert von 1,0 evaluiert, so ist die Regel Nummer 6 die wertvollste, da sie den höchsten Informationsgehalt hat. Regel 6 zeichnet sich durch eine minimale Prämisse und maximale Konklusion aus, was nach [2] Kriterium für nicht-redundante Regeln ist. Regeln 1 bis 5, sowie 7 bis 9 sind somit redundant.

#	**Prämisse**	**Konklusion**
1	Windeln	Bier
2	Windeln	Bier && Chips
3	Windeln	Bier && Flips
4	Windeln && Flips	Bier
5	Windeln && Chips	Bier
6	Windeln	Bier && Chips && Flips
7	Windeln && Chips	Bier && Flips
8	Windeln && Flips	Bier && Chips
9	Windeln && Chips && Flips	Bier

Tabelle 6: Redundante Regeln

GARL kann während des Findens von Regeln überprüfen, ob diese obiges Kriterium für nicht-redundante Regeln erfüllen, und sie andernfalls als Kandidaten verwerfen. Regeln werden dabei als redundante Varianten einer anderen Regel angesehen, wenn sie zusätzlich zu genanntem Kriterium einen gleichen oder niedrigeren Evaluationswert als die Vergleichsregel haben.

Apriori verfügt über keinen Mechanismus, um das Finden von redundanten Regeln zu vermeiden. Es existieren jedoch auch Ansätze für klassische Assoziationslerner, um Regelsätze ohne redundante Regeln zu finden. Am häufigsten werden hierfür closed frequent Itemsets verwendet [6, 8, 9]. Ein frequent Itemset ist closed, wenn es keine Teilmenge eines anderen frequent Itemsets ist, welches genauso oft im Beispieldatensatz vorkommt.

Eine Betrachtung der Redundanz in GARL und Apriori findet in Abschnitt 4.3.3 statt.

3 Optimierungen

Je nach Ausführungsumgebung kann die Implementation des Algorithmus optimiert werden. In diesem Kapitel sollen einige potentielle Verbesserungen diskutiert werden.

3.1 Speichern der Regelbewertung

Der häufigste und rechenaufwendigste Schritt in GARL ist das Evaluieren von Regeln. Da Regeln immer aus der gleichen, leeren Regel erzeugt werden und für Verfeinerungen eventuell identische Beispiele benutzt werden, wiederholen sich die zu bewertenden Regeln sehr häufig.

Es bietet sich darum an, Memoization zu verwenden, indem die Evaluierungsergebnisse zwischengespeichert werden. Die im Rahmen dieser Bachelorarbeit erstellte Implementation nutzt eine Hash Map, um bereits berechnete Ergebnisse schnell abrufbar zu machen. Als Hash dient dabei ein String, der die Attribute und zugehörigen Werte aneinander knüpft und die Prämisse und Konklusion durch ein Sonderzeichen trennt. Damit ist sichergestellt, dass es keine Kollisionen geben kann.

Im Algorithmus wird für eine zu bewertende Regel zuerst geschaut, ob bereits ein Evaluationswert für sie im Hash steht. Wenn ja, wird dieser verwendet, ansonsten wird ein neuer Wert berechnet und in die Hash Map geschrieben.

Die Folge der Verwendung einer Hash Map ist eine signifikant verkürzte Laufzeit bei nur geringfügig gestiegenem Speicherbedarf.

3.2 Multithreading - GARLIC?

Bei modernen Computern gehören Mehrkernprozessoren bereits zum Standard [7], sodass Programme oder Algorithmen, die parallelisiert arbeiten, ein großes Potential für einen Geschwindigkeitszuwachs haben. Die Erweiterung von GARL zu einem Greedy Association Rule Learner Including Concurrency (GARLIC) bietet sich somit an.

In GARL gibt es mehrere Stellen, die für eine gleichzeitige Ausführung geeignet sind. So ließe sich die Bewertung der Verfeinerungen innerhalb von `findBestRule(examples, example)` parallelisieren. Da die Berechnungen unabhängig vom Ergebnis anderer Verfeinerungen sind, sind Synchronisationsprobleme zu erwarten. Wie oben schon erwähnt, ist die Evaluierung von Verfeinerungen der aufwendigste Schritt von GARL; kann dieser Schritt parallel ausgeführt werden, ist ein großer Geschwindigkeitszuwachs zu erwarten.

Algorithmus 3 zeigt die geringfügigen Änderungen der Methode findBestRule (Vergleich siehe Algorithmus 2), die nötig sind, um die Regelbewertung zu parallelisieren. Das Evaluieren der Regeln wird hier in eine einzelne For-Schleife ausgelagert, deren Durchläufe in einzelnen Threads geschehen kann. Diese Form von Multithreading ist besonders für Datensätze mit vielen Attributen geeignet, da in diesem Fall viele Regelkandidaten erstellt werden müssen, die parallel evaluiert werden können.

Die Implementierung kann je nach verwendeter Programmiersprache sehr unterschiedlich aussehen. Die im Rahmen dieser Bachelorarbeit erstellte Implementation benutzt obige Parallelisierung. Im folgenden wird mit GARLIC die Multithreading Implementation des Algorithmus bezeichnet. Die Auswirkung der Parallelisierung auf die Laufzeit wird in 4.6.1 diskutiert.

Eine weitere Möglichkeit den Algorithmus zu parallelisieren besteht in der äußeren Schleife, gezeigt in Algorithmus 1. Das komplette Finden von Regeln könnte gleichzeitig geschehen. Jedoch gibt es dabei mehrere potentielle Probleme, die behandelt werden müssten. So kann es geschehen, dass in parallel arbeitenden Threads das gleiche Beispiel zur Regel-Erstellung ausgewählt wird, womit die Parallelisierung nutzlos würde.

Zudem kann es ohne weitere Anpassungen passieren, dass die geforderte coverageNumber, die minimale Anzahl mit der jedes Beispiel abgedeckt werden muss, von allen Beispielen überschritten wird. Somit würden mehr Regeln gelernt werden, als für die gegebene coverageNumber nötig wären.

Algorithmus 3 Pseudocode der modifizierten Methode findBestRule(examples, example)

```
1:  currentBestRule = emptyRule()
2:  overallBestRule = currentBestRule
3:  evaluate(currentBestRule)
4:  rules = {currentBestRule}
5:  noImprovementCounter = 0;
6:  while rules ≠ ϕ
    && isImprovementPossible(overallBestRule)
    && noImprovementCounter < maxNoImprovement do
7:    newOverallBestRule = false
8:    candidates = selectCandiates(rules)
9:    rules = rules \ candidates
10:   for candidate ϵ candidates do
11:     refinements = refineRule(candidate, examples, example)
12:     refinements = refinements \ theory
13:     {Multithreaded For-Loop}
14:     for refinement ϵ refinements do
15:       evaluate(refinement)
16:     end for
17:     for refinement ϵ refinements do
18:       if !isRefinementRedundant(refinement) then
19:         rules = rules ∪ refinement
20:         if (refinement > currentBestRule) then
21:           currentBestRule = refinement
22:           if (refinement > overallBestRule) then
23:             overallBestRule = refinement
24:             newOverallBestRule = true
25:           end if
26:         end if
27:       end if
28:     end for
29:   end for
30:   rules = filterRules(rules)
31:   currentBestRule = emptyRule()
32:   evaluate(curentBestRule)
33:   if !newOverallBestRule then
34:     noImprovementCounter = noImprovementCounter + 1
35:   end if
36: end while
37: return overallBestRule
```

4 Evaluation

Dieses Kapitel dient der Bewertung des vorgestellten Algorithmus. Dazu werden die Ergebnisse mit einer Apriori Implementation verglichen. In Betracht gezogen wird sowohl die Qualität der gefundenen Regeln als auch die Laufzeit.

4.1 Weka

Zum Vergleich wird eine Apriori Implementation der Software Weka [5] verwendet. Weka (Waikato Environment for Knowledge Analysis) ist eine Open Source Software, die eine Ansammlung von Lernalgorithmen bereitstellt. Die Implementation von GARLIC bedient sich einiger weniger Weka Klassen zum Laden und Verwalten von Datensätzen.

4.2 Vergleich

Für die folgenden Tests wurde ein Datensatz benutzt, der Informationen über die Passagiere der Titanic-Katastrophe beinhaltet.

Für alle 2201 Passagiere und Crewmitglieder der Titanic liegen folgende Attribute mit den dazugehörigen möglichen Werten vor:

- **Class** - {1st, 2nd, 3rd, crew}
- **Age** - {adult, child}
- **sex** - {male, female}
- **survived** - {yes, no}

Der Datensatz wird üblicherweise eher im Zusammenhang mit Klassifikationen verwendet, kann aber ebenso für Assoziationslerner eingesetzt werden.

Zum Vergleich mit Wekas Apriori Implementation wird als Evaluationsmetrik Confidence verwendet, welche sowohl in GARLIC als auch Weka implementiert ist. Beispielhafte Ergebnisse für andere Evaluationsmetriken finden sich im Anhang.

4.3 Qualität der Regeln

4.3.1 Ergebnisse

Die Ergebnisse GARLICs überprüfen, falls nicht anders angegeben, alle Regeln auf Redundanz. Eine Diskussion über redundante Regeln in GARLIC findet in Abschnitt 4.3.3 statt.

Tabelle 7 zeigt das Ergebnis eines Durchlaufs von GARLIC mit Confidence als Evaluator und coverageNumber = 1. Jedes Beispiel im Datensatz muss somit nur einmal abgedeckt werden.

#	Prämisse	p + n	Konklusion	p	Confidence
1	class == 1st && survived == no	122	age == adult	122	1,00
2	class == crew	885	age == adult	885	1,00
3	class == crew && survived == no	673	sex == male	670	1,00
4	class == 1st && sex == female	145	age == adult	144	0,99
5	class == 1st	325	age == adult	319	0,98
6	sex == male && survived == no	1364	age == adult	1329	0,97
7	class == crew	885	sex == male	862	0,97
8	survived == no	1490	age == adult	1438	0,97
9	survived == yes	711	age == adult	654	0,92

Tabelle 7: Ergebnis GARLIC für coverageNumber = 1 und minimumEvaluation = 0,5

Die gefundenen Regeln teilen sich bis auf zwei Ausnahmen die Konklusion `age == adult`. Das liegt an der ungleichen Verteilung von Erwachsenen (2092) und Kindern (109) auf der Titanic. Für mehr Ergebnisse mit unterschiedlicherem Charakter muss die coverageNumber erhöht werden, sodass jedes Beispiel mehr als einmal abgedeckt werden muss.

Zusätzlich zur coverageNumber hat aber auch der Random Seed-Wert einen großen Einfluss auf das Ergebnis. Da eine gefundene Regel immer aus den Elementen des gewählten Beispiels besteht, führen Beispiele mit anderen Werten meistens auch zu unterschiedlichen Regeln. Tabelle 8 zeigt, wie stark die Anzahl der gefundenen Regeln bei unterschiedlichen

Werten für coverageNumber und randomSeed schwankt. Angegeben ist neben der Anzahl der gefundenen Regeln auch der Durchschnittswert für fünf verschiedene Seed-Werte sowie die maximale Schwankung um diesen Durchschnittswert in Prozent.

	Seed = 1	**Seed = 2**	**Seed = 3**	**Seed = 4**	**Seed = 5**	**Durchschnitt**	**Schwankung**
CN = 1	6	3	7	6	9	6,2	51,62%
CN = 2	12	7	14	12	15	20,8	41,67%
CN = 3	17	21	21	18	16	18,6	11,43%
CN = 4	30	30	28	22	32	28,4	22,54%
CN = 5	33	30	28	26	33	30,0	13,33%
CN = 10	39	34	38	36	35	53,0	6,67%
CN = 1000	50	44	47	44	51	73,0	6,78%

Tabelle 8: Anzahl Ergebnisse für verschiedene coverageNumber und randomSeed Werte

Zu sehen ist, wie sich die Schwankung mit höherer Coverage Number verringert. Da sich die Laufzeit des Algorithmus mit steigender CoverageNumber erhöht, sind extreme Werte aber nur bedingt zu empfehlen.

4.3.2 Apriori

Apriori findet für einen minimalen Support und eine minimale Confidence alle möglichen Regeln. Es werden also alle Itemsets verworfen, die nicht oft genug vorkommen, um den minimalen Support zu erfüllen. GARLIC findet hingegen Regeln mit beliebig geringem Support. Für einen besseren Vergleich zwischen GARLIC und Apriori wurde in folgenden Beispielen der minimum Support so niedrig gewählt, dass auch Itemsets, die nur einmal vorkommen, beim Finden von Regeln berücksichtigt werden. Die Laufzeit von Apriori erhöht sich durch diese Maßnahme. Mehr dazu in Abschnitt 4.6

4.3.3 Redundanz

Tabelle 9 zeigt die acht Regeln mit dem höchsten Evaluationswert bei einem Durchlauf von Apriori. Insgesamt findet Apriori 178 Regeln.

#	**Prämisse**	**p + n**	**Konklusion**	**p**	**Confidence**
1	class == crew	885	age == adult	885	1,00
2	class == crew && sex == male	862	age == adult	862	1,00
3	class == crew && survived == no	673	age == adult	673	1,00
4	class == crew && sex == male && survived == no	670	age == adult	670	1,00
5	class == crew && survived == yes	212	sex == male	212	1,00
6	class == crew && sex == male && survived == yes	192	age == adult	192	1,00
7	class == 2nd && survived == no	167	age == adult	167	1,00
8	class == 2nd && sex == male && survived == no	154	age == adult	154	1,00

Tabelle 9: Ergebnis Apriori für minimum Confidence = 0,8

Bei Betrachtung der ersten beiden Regeln fällt jedoch auf, dass sie nahezu identisch sind. Beide Regeln haben den „perfekten" Evaluationswert von 1,00 und unterscheiden sich lediglich durch ein zusätzliches Attribut in der Prämisse der zweiten Regel.

!!!!! Anschaulich dürfte klar sein, dass aus der Regel, dass alle Crewmitglieder erwachsen sind, auch hervorgeht, dass alle männlichen Crewmitglieder erwachsen sind.!!!!!!

Die zweite Regel bietet somit keinen Mehrwert gegenüber der ersten und kann als überflüssig angesehen werden. Gleiches gilt für die dritte Regel, die eine weitere Spezialisierung der zweiten und somit auch der ersten Regel darstellt. Auch Regeln vier bis sechs sind weitere Verfeinerungen der ersten Regel, welche aber bereits perfekt ist. Erst die siebte Regel ist wieder nützlich und stellt einen echten Mehrwert dar. Die achte Regel ist wiederum eine Spezialisierung der siebten.

Solche Verfeinerungen ohne verbesserten Evaluationswert finden sich in den Ergebnissen Aprioris häufig. Von 178 gefundenen Regeln sind 85 redundant nach dem Kriterium aus Abschnitt 2.5. Es bietet sich an, die redundanten Regeln aus dem Ergebnis herauszufiltern, was jedoch weiteren Aufwand bedeutet.

GARLIC verwirft Regelkandidaten, die bereits Teil der Theorie sind. Dadurch werden deutlich weniger redundante Regeln gefunden. Unmöglich ist das Finden von redundanten Regeln bei ausgeschalteter Redundanzüberprüfung jedoch nicht. Angenommen, in einem ersten Durchlauf von findBestRule() werden aus dem Beispiel

$$class == crew, age == adult, sex == male, survived == no$$

folgende Regelkandidaten am höchsten bzw. zweithöchsten bewertet:

$$class == crew \Rightarrow age == adult, Confidence : 1,00$$

$$sex == male \Rightarrow age == adult, Confidence : 0,96$$

Der erste Kandidat wird zur Theorie hinzugefügt. Der Algorithmus findet mit Hilfe von anderen Beispielen weitere Regeln, so dass alle Beispiele mindestens einmal abgedeckt werden. Wird vom Algorithmus gefordert, dass er jedes Beispiel mehr als einmal abgedeckt wird, könnte er in einem neuen Durchlauf von findBestRule() wiederum das Beispiel

$$class == crew, age == adult, sex == male, survived == no$$

auswählen und wieder folgende Regelkandidaten erstellen:

$$class == crew \Rightarrow age == adult$$

$$sex == male \Rightarrow age == adult$$

Der erste Kandidat entspricht der zuerst gefundenen Regel und wird gestrichen, da er bereits Teil der Theorie ist. Nun wird der zweite Kandidat zu folgender Regel verfeinert:

$$class == crew \&\& sex == male \Rightarrow age == adult, Confidence : 1,00$$

Diese Regel wird nun der Theorie hinzugefügt, stellt aber eine redundante Variante der ersten Regel dar.

In der Praxis tritt ein solcher Fall für einen maxNoImprovement Wert von 0 recht selten ein. Bei einem Durchlauf mit coverageNumber = 5 findet GARLIC ohne Redundanzüberprüfung 38 Regeln, deren zehn am höchsten bewertete Regeln in Tabelle 10 gezeigt werden. Redundante Regeln sind fett gedruckt. Insgesamt gibt es im Ergebnis sechs redundante Regeln, die allerdings nicht alle zu den zehn am höchsten bewerteten gehören.

#	Prämisse	p + n	Konklusion	p	Confidence
1	age == child && survived == no	52	class == 3rd	52	1
2	**class == crew && sex == female && survived == yes**	**20**	**age == adult**	**20**	**1**
3	**class == crew && sex == female**	**23**	**age == adult**	**23**	**1**
4	class == 2nd && age == child && sex == female	13	survived == yes	13	1
5	**class == crew && sex == male**	**862**	**age == adult**	**862**	**1**
6	class == 1st && survived == no	122	age == adult	122	1
7	class == crew	885	age == adult	885	1
8	**class == crew && survived == no**	**673**	**sex == male**	**670**	**1**
9	class == 1st && sex == female	145	age == adult	144	0,99
10	class == 1st	325	age == adult	319	0,98

Tabelle 10: Ergebnis GARLIC für coverageNumber = 5 ohne Redundanzüberprüfung

Für maxNoImprovement Werte größer 0 erhöht sich die Anzahl der redundanten Regeln. Da nur wenige gute zusätzliche Regeln gefunden werden, gleichzeitig aber die Laufzeit steigt, empfiehlt sich für maxNoImprovement ein Wert von 0.

Um redundante Regeln ganz zu vermeiden, filtert GARLIC in der Voreinstellung automatisch redundante Regeln heraus. Da Regelkandidaten dabei nur mit bereits gelernten Regeln verglichen werden, ist dies effizienter als die nachträgliche Filterung von einem kompletten Regeldatensatz.

#	Prämisse	p + n	Konklusion	p	Confidence
1	age == child && survived == no	52	class == 3rd	52	1,00
2	class == 1st && survived == no	122	age == adult	122	1,00
3	class == crew	885	age == adult	885	1,00
4	class == crew && survived == no	673	sex == male	670	1,00
5	class == 1st && sex == female	145	age == adult	144	0,99
6	class == 1st	325	age == adult	319	0,98
7	sex == male && survived == no	1364	age == adult	1329	0,97
8	class == crew	885	sex == male	862	0,97
9	survived == no	1490	age == adult	1438	0,97
10	sex == male	1731	age == adult	1667	0,96

Tabelle 11: Ergebnis GARLIC für coverageNumber = 5

#	Prämisse	p + n	Konklusion	p	Confidence
1	class == crew	885	age == adult	885	1,00
2	class == 2nd && survived == no	167	age == adult	167	1,00
3	class == 1st && survived == no	122	age == adult	122	1,00
4	age == child && survived == no	52	class == 3rd	52	1,00
5	class == 2nd && age == child	24	survived == yes	24	1,00
6	class == 1st && age == child	6	survived == yes	6	1,00
7	class == crew && survived == no	673	sex == male	670	1,00
8	class == 1st && sex == female	145	age == adult	144	0,99
9	class == 1st	325	age == adult	319	0,98
10	sex == male && survived == no	1364	age == adult	1329	0,97

Tabelle 12: Ergebnis Apriori ohne redundante Regeln

4.3.4 Vergleich

Für den folgenden direkten Vergleich wurde das Ergebnis Aprioris händisch um alle redundanten Regeln bereinigt. Tabellen 11 und 12 zeigen die zehn Regeln mit höchstem Confidence Wert. GARLIC arbeitet mit einem coverageNumber Wert von fünf.

Die Ergebnisse der beiden Algorithmen fallen sehr unterschiedlich aus. Während Apriori Regeln mit höherem Confidence Wert findet, decken GARLICs Regeln mehr Beispiele ab. Im Durchschnitt gibt es für die ersten zehn GARLIC Regeln 748,8 positive Beispiele, für die ersten zehn Apriori Regeln nur 371,8. Zu beachten ist, dass alle von GARLIC gefundenen Regeln auch von Apriori gefunden werden, dort jedoch hinter Regeln mit höherem Confidence Wert liegen. Berücksichtigt man, dass Confidence für Regeln mit geringer Abdeckung und nur positiven Beispielen hohe Werte liefert, kann man die Ergebnisse GARLICs insofern als nützlicher interpretieren, als dass sich der Algorithmus auf relevantere Regeln „konzentriert". Ursächlich ist die Wahl eines Beispiels zur Regelerstellung. Da dies zufällig geschieht, ist es wahrscheinlich, dass das gewählte Beispiel häufig vorkommende Itemsets enthält und es somit viele weitere ähnliche oder gleiche Beispiele im Datensatz gibt. Gleichzeitig ist aber nicht garantiert, dass GARLIC entsprechende Itemsets wählt.

Nutzt man statt Confidence die Evaluationsmetrik Laplace (siehe Abschnitt 2.4), stellt man fest, dass sich das Ergebnis für GARLIC nur geringfügig verändert, während die Reihenfolge der Apriori Regeln größeren Schwankungen unterworfen ist. Tabellen 13 und 14 zeigen die zehn Regeln mit höchstem Laplace Wert. Da Weka keine Laplace Implementation für Apriori bietet, wurden die Evaluationswerte manuell ausgerechnet.

Während GARLICs Ergebnis nur ein wenig anders sortiert ist, verschiebt sich das Ergebnis Aprioris zugunsten von Regeln, die mehr Beispiele abdecken. Sehr selten abgedeckte Regeln finden sich nicht mehr in der Liste der ersten zehn wieder. Dennoch ist auch mit einer Laplace Evaluation die durchschnittliche positive Abdeckung der ersten zehn GARLIC Regeln mit 748,8 deutlich höher als die der ersten zehn Apriori Regeln mit 472,5.

#	Prämisse	p + n	Konklusion	p	Laplace
1	class == crew	885	age == adult	885	1,00
2	class == crew && survived == no	673	sex == male	670	0,99
3	class == 1st && survived == no	122	age == adult	122	0,99
4	class == 1st && sex == female	145	age == adult	144	0,99
5	age == child && survived == no	52	class == 3rd	52	0,98
6	class == 1st	325	age == adult	319	0,98
7	sex == male && survived == no	1364	age == adult	1329	0,97
8	class == crew	885	sex == male	862	0,97
9	survived == no	1490	age == adult	1438	0,96
10	sex == male	1731	age == adult	1667	0,96

Tabelle 13: Ergebnis GARLIC für coverageNumber = 5 - Laplace

#	Prämisse	p + n	Konklusion	p	Laplace
1	class == crew	885	age == adult	885	1,00
2	class == 2nd && survived == no	167	age == adult	167	0,99
3	class == crew && survived == no	673	sex == male	670	0,99
4	class == 1st && survived == no	122	age == adult	122	0,99
5	class == 1st && sex == female	145	age == adult	144	0,99
6	age == child && survived == no	52	class == 3rd	52	0,98
7	class == 1st	325	age == adult	319	0,98
8	sex == male && survived == no	1364	age == adult	1329	0,97
9	class == crew	885	sex == male	862	0,97
10	class == 1st && sex == male	180	age == adult	175	0,97

Tabelle 14: Ergebnis Apriori ohne redundante Regeln - Laplace

4.4 Abdeckung

Tabelle 15 zeigt, von wie vielen Regeln die Beispiele des Datensatzes im Durchschnitt für verschiedene CoverageNumber und randomSeed Werte abgedeckt werden. Beispiele werden von einer Regel abgedeckt, wenn die Prämisse der Regel Teil des Beispiels ist.

	Seed = 1	**Seed = 2**	**Seed = 3**	**Seed = 4**	**Seed = 5**	**Durchschnitt**
CN = 1	2,61	1,16	2,88	1,58	3,00	2,25
CN = 2	5,48	3,17	7,81	4,84	6,71	5,60
CN = 3	7,02	10,51	10,93	7,97	6,92	8,67
CN = 4	19,01	23,26	22,98	9,69	25,07	20,00
CN = 5	28,18	42,64	24,88	11,41	38,84	29,19
CN = 10	52,29	84,42	50,71	31,00	52,33	54,15
CN = 1000	3898,54	3732,26	3780,06	3709,05	3866,99	3797,38

Tabelle 15: Durchschnittliche Abdeckung der Beispiele durch alle gefundenen Regeln

Deutlich zu sehen ist, wie stark die durchschnittliche Abdeckung mit größer werdender CoverageNumber zunimmt. Im Vergleich mit Tabelle 8 scheinen die Beispiele bei hohen CoverageNumber Werten von mehr Regeln abgedeckt zu werden, als überhaupt gefunden wurden. Allerdings werden nur Regeln mit einem minimalen Evaluationswert (hier 0.5) zur Theorie hinzugefügt, die entsprechenden Beispiele aber dennoch als abgedeckt markiert, um eine Terminierung des Algorithmus für beliebig große CoverageNumber Werte zu gewährleisten.

Tabelle 16 zeigt ebenfalls, wie häufig die Beispiele abgedeckt werden. Es werden hier jedoch nur die Regeln der Theorie, also solchen mit einem minimalen Evaluationswert, berücksichtigt.

	Seed = 1	**Seed = 2**	**Seed = 3**	**Seed = 4**	**Seed = 5**	**Durchschnitt**
CN = 1	2,61	1,16	2,88	1,58	3,00	2,25
CN = 2	5,48	3,17	7,81	4,84	6,71	5,60
CN = 3	7,02	10,51	10,93	7,97	6,92	8,67
CN = 4	11,89	12,41	12,19	9,69	12,55	11,75
CN = 5	12,13	12,41	12,19	11,41	12,76	12,18
CN = 10	12,67	12,62	12,87	12,92	12,87	12,79
CN = 1000	13,27	13,37	13,50	13,19	14,12	13,49

Tabelle 16: Durchschnittliche Abdeckung der Beispiele durch Regeln mit minimalem Confidence Wert

Besonders bei höheren CoverageNumber Werten weicht die Abdeckung durch Regeln der Theorie von der Abdeckung durch alle Regeln erheblich ab.

Apriori deckt mit 178 gefundenen Regeln jedes Beispiel durchschnittlich 32,21-mal ab. Filtert man die redundanten Regeln heraus, decken die verbleibendenen 93 Regeln die Beispiele im Durchschnitt noch 22,30-mal ab. GARLIC ermöglicht es somit auf einfache Weise, die Beispiele durch eine geringere Anzahl an Regeln zu erklären, welche dafür aussagekräftiger sind.

4.5 Anzahl Attribute in der Konklusion

Auffällig ist, dass alle gefundenen Regeln lediglich ein Attribut in der Konklusion haben. Das erinnert im ersten Moment an Klassifikationsregeln, diese haben jedoch ein festes Attribut als Konklusion. Der Grund für das Fehlen von Konklusionen mit mehr als einem Attribut liegt in der verwendeten Evaluationsmetrik. Mit Confidence oder Laplace evaluierte Regeln können bei steigender Anzahl an Attributen in der Konklusion höchstens den gleichen Evaluationswert, aber niemals einen größeren erhalten.

Beweis. Sei $\frac{p}{p+n}$ die Confidence-Evaluation einer Regel R und $\frac{p'}{p'+n'}$ die Confidence-Evaluation der Regel R', welche eine Spezialisierung von R durch Hinzufügen eines Attributs in der Konklusion darstellt.

Zu Zeigen ist:

$$\frac{p'}{p'+n'} <= \frac{p}{p+n}$$

Offensichtlich gilt:

$$p' <= p, n' >= n$$

Wobei n' genauso viel größer wird wie p' kleiner, also:

$$p - p' = n' - n$$

Oder:

$$p' = p - n' + n$$

Durch Einsetzen in $\frac{p'}{p'+n'}$ erhalten wir:

$$\frac{p - n' + n}{p - n' + n + n'}$$

Oder:

$$\frac{p - n' + n}{p + n}$$

Aus $n' >= n$ folgt $-n' + n <= 0$ und somit auch:

$$\frac{p - n' + n}{p + n} <= \frac{p}{p + n}$$

□

Der Beweis für die Evaluationsmetrik Laplace funktioniert analog.

Apriori findet Regeln mit mehr als einem Attribut in der Konklusion. Diese haben aber einen niedrigen Evaluationswert und auch einen niedrigen Rang oder liegen unter dem minimumEvaluation Wert. Da in GARLIC aufgrund der hill-climbing Strategie nur Verfeinerungen mit größerem Evaluationswert betrachtet werden, kann es mit Confidence und Laplace Evaluation nur genau ein Attribut in der Konklusion geben.

Um Regeln mit mehr als einem Attribut in der Konklusion zu erhalten, muss eine andere Evaluationsmetrik gewählt werden. Die in GARLIC implementierten Metriken Weighted Relative Accuracy, Lift und Correlation führen alle zu Regeln mit mehreren Attributen in der Konklusion. Beispielhafte Ergebnisse finden sich im Anhang, werden jedoch nicht weiter diskutiert, da Wekas Apriori die Metriken nicht unterstützt und somit ein Vergleich schwierig ist.

4.6 Laufzeit

Zu prüfen ist, ob die Laufzeit des GARLIC Algorithmus im Vergleich zu Apriori niedrig genug ist, um ihn praxistauglich zu machen. Verglichen wird die Performanz der GARLIC Implementation und Wekas Apriori Implementation.

Zum Testen der Laufzeit wird ein anderer, größerer Datensatz verwendet. Der Datensatz namens „Kropt" enthält Daten über die Position von Schachfiguren. Genauer gesagt, von einem schwarzen König, einem weißen König und einem weißen Bauer. Die Beispieldaten geben an, wie viele Züge der weiße Spieler braucht, um den schwarzen zu schlagen. Der Datensatz verfügt über 28056 Beispiele mit sieben Attributen, die bis zu 16 verschiedene Werte annehmen können. Tabelle 17 zeigt in Sekunden, wie lange GARLIC für die Analyse des oben genannten Datensatz bei verschiedenen coverageNumber und Seed Werten benötigt. Der maxNoImprovement Wert ist für alle Tests null. Auffällig ist, dass sich die Laufzeit für größere coverageNumber Werte nur geringfügig erhöht. Das liegt an der verwendeten Hashmap. Regelevaluierungen wiederholen sich gerade bei hohen coverageNumber Werten sehr häufig, so dass das Hashen der Ergebnisse einen deutlichen Geschwindigkeitszuwachs bedeutet.

	Seed = 1	**Seed = 2**	**Seed = 3**	**Seed = 4**	**Seed = 5**	**Durchschnitt**
CN = 1	153	157	250	200	125	177
CN = 2	210	228	262	213	208	224,2
CN = 3	261	250	274	216	227	245,6
CN = 4	270	276	280	235	237	259,6
CN = 5	278	341	307	261	271	291,6

Tabelle 17: Laufzeit GARLIC in Sekunden - maxThreads = 1 - maxNoImprovement = 0

Wie bereits erwähnt, benötigt Apriori die Angabe eines Minimum Supports, so dass der Algorithmus nur Regeln in Betracht zieht, dessen Itemsets häufig genug vorkommen. Die bisherigen Tests setzten den Minimum Support so niedrig, dass eine Häufigkeit von eins ausreichte. Für den nun vorliegenden Datensatz stößt Apriori bei extrem niedrigem Support jedoch an seine Grenzen. Für einen Support von nur einem Beispiel war es beim Testen nicht möglich, Wekas Implementation terminieren zu lassen. Mehrere Stunden Laufzeit und ein in die Gigabyte gehender Speicherbedarf zeigen, dass Apriori auf einen höheren Support angewiesen ist.

Minimum Support	28	281	1403
Minimum Confidence	0.5	0.5	0.5
Laufzeit in Sekunden	98	31	7
Speicherbedarf	105 Megabyte	101 Megabyte	81 Megabyte
Gefundene Regeln	7937	207	11

Tabelle 18: Performanz Apriori

Wekas Apriori Implementation benötigt eine Minimum Support Angabe als Prozentsatz der Anzahl der Beispiele. Die in Tabelle 18 verwendeten Werte entsprechen 0,001%, 0,01% und 0,05%. Mit einem Wert niedriger 0,001% terminierte der Algorithmus nicht. Die Wahl von einem höheren Minimum Support Wert beschleunigt Apriori, führt aber zu deutlich weniger gefundenen Regeln.

4.6.1 Multithreading

Die zu dieser Bachelorarbeit gehörende GARLIC Implementation ist Multithreaded (siehe Abschnitt 3.2). Tabelle 19 zeigt, wie sich die Verwendung von mehreren Threads auf die Laufzeit des Algorithmus auswirkt. Außerdem angegeben ist der durchschnittliche Geschwindigkeitszuwachs für die jeweilige Threadanzahl. Die Tests wurden auf einem Computer mit acht Prozessoren ausgeführt, so dass jeder Thread vom Betriebssystem einen eigenen Prozessor zugewiesen bekommen könnte.

	Threads = 1	**Threads = 2**	**Threads = 3**	**Threads = 4**	**Threads = 5**
CN = 1	153	86	71	59	57
CN = 2	210	127	107	93	89
CN = 3	261	163	138	121	118
CN = 4	270	169	142	125	123
CN = 5	278	175	148	128	126
Beschleunigung	1,00x	1,64x	1,96x	2,2x	2,33x

Tabelle 19: Laufzeit GARLIC in Sekunden - Seed = 1 - maxNoImprovement = 0

Die Verwendung von Multithreading bringt einen deutlichen Geschwindigkeitszuwachs. Obwohl nicht alle Teile der GARLIC Implementation von mehreren Prozessoren profitieren, lässt sich die Geschwindigkeit alleine durch die parallelisierte Evaluation mehr als verdoppeln. Deutlich zu sehen ist, dass die Geschwindigkeit bei mehr als vier Evaluationsthreads nur noch marginal ansteigt. Hier würde GARLIC von der Parallelisierung weiterer Programmabschnitte profitieren. Der Vorteil der Parallelisierung lässt mit zunehmenden CoverageNumber Werten leicht nach, da gleichzeitig immer mehr Regelevaluationen aus dem Hash entnommen werden, was durch Multithreading kaum beschleunigt werden kann.

5 Fazit

Ein großes Problem klassischer Assoziationslerner wie Apriori ist die Anzahl der gefundenen Regeln. Sie kann bei großen Datensätzen schnell unhandlich werden. Besonders gravierend dabei ist es, dass viele der gefundenen Regeln redundante Varianten von anderen Regeln sind und somit keinen praktischen Nutzen für den Anwender haben.

Der in dieser Bachelorarbeit vorgestellte Greedy Association Rule Learner Including Concurrency (GARLIC) sorgt mit einem von Klassifizierern entlehntem Ansatz dafür, dass weniger Regeln gefunden werden, die dafür für den Benutzer interessanter sein dürften. Redundante Regeln werden deutlich seltener gefunden und können mit einem einfachen Filter noch während des Lernens entfernt werden.

Der Verzicht auf einen minimum Support bedeutet, dass GARLIC keine Regeln von vornherein ausschließt. In Datensätzen, in denen es überwiegend Regeln mit geringem Support gibt, findet GARLIC entsprechende Regeln. Hat ein Datensatz größtenteils Regeln mit hohem Support, findet GARLIC diese Regeln. Der Benutzer muss somit keine Informationen über den Datensatz haben, um nützliche Ergebnisse zu bekommen. Die Wahl eines geeigneten minimum Support für klassische Assoziationslerner ist nicht immer einfach, da der Benutzer im schlechtesten Fall keine Informationen über den Datensatz und somit keine Erwartungshaltung an die zu findenden Regeln hat.

GARLIC benötigt die Angabe eines minimum Coverage Wertes, der angibt, wie oft jedes Beispiel abgedeckt werden soll. Das bedeutet zwar einen gewissen Mehraufwand für den Benutzer, erlaubt dem Benutzer aber eine Möglichkeit, Einfluss auf die Anzahl der zu findenden Regeln zu nehmen, ohne dass er Informationen über den Datensatz benötigt.

Die Laufzeit der aktuellen GARLIC Implementation ist noch recht hoch, kommt aber in einer Mehrprozessor-Umgebung zumindest an den Apriori Algorithmus heran, wenn dieser einen niedrigen minimum Support verwendet. Berücksichtigt man eine Filterung von redundanten Regeln aus dem Ergebnis Aprioris sowie den eventuellen Bedarf an mehreren Versuchen zum Wählen eines geeigneten minimum Support Wert, ist der Performanzvorsprung Aprioris nur noch bedingt gegeben. Weitere Optimierungen der GARLIC Implementation könnten noch zusätzliche Geschwindigkeitsgewinne nach sich ziehen.

Der Einsatz GARLICs gegenüber anderen Assoziationslernern kann somit besonders in Umgebungen empfohlen werden, in denen mehrere Prozessoren zur Verfügung stehen und besonderer Wert auf eine überschaubare Anzahl von Regeln hoher Qualität gelegt wird.

Literatur

[1] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. pages 307–328, 1996.

[2] Yves Bastide, Nicolas Pasquier, Rafik Taouil, Gerd Stumme, and Lotfi Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. In *CL '00: Proceedings of the First International Conference on Computational Logic*, pages 972–986, London, UK, 2000. Springer-Verlag.

[3] Johannes Fürnkranz. Seperate-and-conquer rule learning. *Artificial Intelligence Review*, 1999.

[4] B. Goethals. Survey on frequent pattern mining. Manuscript, 2003.

[5] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[6] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 398–416, London, UK, 1999. Springer-Verlag.

[7] Valve Software. Steam hardware survey. http://store.steampowered.com/hwsurvey, January 2010.

[8] Ruili Wang, Luofeng Xu, Stephen Marsland, and Ramesh Rayudu. An efficient algorithm for mining frequent closed itemsets in dynamic transaction databases. *Int. J. Intell. Syst. Technol. Appl.*, 4(3/4):313–326, 2008.

[9] Mohammed J. Zaki. Generating non-redundant association rules. Technical report, Computer Science Department, Rensselaer Polytechnic Institute, Troy NY 12180.

[10] Mohammed J. Zaki. Mining non-redundant association rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004.

A Anhang

A.1 Ergebnisse verschiedener Evaluationsmetriken

An dieser Stelle folgt eine beispielhafte Auflistung von Testergebnissen mit allen in GARLIC implementierten Evaluationsmetriken. In allen Beispielen wurde der Titanic Datensatz (siehe Kapitel 4.2) verwendet. Als Random-Seed Wert wurde jeweile 5 gewählt, während für -maxNoImprovement immer 0 verwendet wurde. Der minimale Evaluierungswert variiert, um für jede Metrik eine Menge von Regeln zu finden.

#	Prämisse	p + n	Konklusion	p	Confidence
1	age == child && survived == no	52	class == 3rd	52	1
2	class == 1st && survived == no	122	age == adult	122	1
3	class == crew	885	age == adult	885	1
4	class == crew && survived == no	673	sex == male	670	1
5	class == 1st && sex == female	145	age == adult	144	0,99
6	class == 1st	325	age == adult	319	0,98
7	sex == male && survived == no	1364	age == adult	1329	0,97
8	class == crew	885	sex == male	862	0,97
9	survived == no	1490	age == adult	1438	0,97
10	sex == male	1731	age == adult	1667	0,96
11	age == adult && survived == no	1438	sex == male	1329	0,92
12	survived == yes	711	age == adult	654	0,92
13	survived == no	1490	sex == male	1364	0,92
14	sex == female	470	age == adult	425	0,9
15	class == 3rd	706	age == adult	627	0,89
16	class == crew && sex == female	23	survived == yes	20	0,87
17	sex == female && survived == no	126	class == 3rd	106	0,84
18	class == 3rd && age == adult && sex == male	462	survived == no	387	0,84
19	class == 3rd && sex == male	510	survived == no	422	0,83
20	age == adult && sex == male	1667	survived == no	1329	0,8
21	age == adult	2092	sex == male	1667	0,8
22	sex == male	1731	survived == no	1364	0,79
23	class == crew	885	survived == no	673	0,76
24	class == 3rd && age == adult	627	survived == no	476	0,76
25	class == 3rd	706	survived == no	528	0,75
26	age == adult && sex == female	425	survived == yes	316	0,74
27	sex == female	470	survived == yes	344	0,73
28	class == 3rd	706	sex == male	510	0,72
29	age == adult	2092	survived == no	1438	0,69
30	age == adult && sex == male && survived == yes	338	class == crew	192	0,57
31	sex == male && survived == yes	367	class == crew	192	0,52
32	age == adult && sex == male	1667	class == crew	862	0,52
33	survived == yes	711	sex == male	367	0,52

Tabelle 20: Confidence - minimumEvaluation = 0.5

#	Prämisse	p + n	Konklusion	p	Laplace
1	class == crew	885	age == adult	885	1
2	class == crew && survived == no	673	sex == male	670	0,99
3	class == 1st && survived == no	122	age == adult	122	0,99
4	class == 1st && sex == female	145	age == adult	144	0,99
5	age == child && survived == no	52	class == 3rd	52	0,98
6	class == 1st	325	age == adult	319	0,98
7	sex == male && survived == no	1364	age == adult	1329	0,97
8	class == crew	885	sex == male	862	0,97
9	survived == no	1490	age == adult	1438	0,96
10	sex == male	1731	age == adult	1667	0,96
11	age == adult && survived == no	1438	sex == male	1329	0,92
12	survived == yes	711	age == adult	654	0,92
13	survived == no	1490	sex == male	1364	0,91
14	sex == female	470	age == adult	425	0,9
15	class == 3rd	706	age == adult	627	0,89
16	class == crew && sex == female	23	survived == yes	20	0,84
17	class == 3rd && age == adult && sex == male	462	survived == no	387	0,84
18	sex == female && survived == no	126	class == 3rd	106	0,84
19	class == 3rd && sex == male	510	survived == no	422	0,83
20	age == adult && sex == male	1667	survived == no	1329	0,8
21	age == adult	2092	sex == male	1667	0,8
22	sex == male	1731	survived == no	1364	0,79
23	class == crew	885	survived == no	673	0,76
24	class == 3rd && age == adult	627	survived == no	476	0,76
25	class == 3rd	706	survived == no	528	0,75
26	age == adult && sex == female	425	survived == yes	316	0,74
27	sex == female	470	survived == yes	344	0,73
28	class == 3rd	706	sex == male	510	0,72
29	age == adult	2092	survived == no	1438	0,69
30	age == adult && sex == male && survived == yes	338	class == crew	192	0,57
31	sex == male && survived == yes	367	class == crew	192	0,52
32	age == adult && sex == male	1667	class == crew	862	0,52
33	survived == yes	711	sex == male	367	0,52

Tabelle 21: Laplace - minimumEvaluation = 0.5

#	Prämisse	p + n	Konklusion	p	WRA
1	age == adult && sex == male	1667	survived == no	1329	0,09
2	survived == no	1490	age == adult && sex == male	1329	0,09
3	sex == male	1731	age == adult && survived == no	1329	0,09
4	survived == yes	711	sex == female	344	0,09
5	sex == female	470	survived == yes	344	0,09
6	class == crew	885	age == adult && sex == male	862	0,09
7	age == adult && sex == male	1667	class == crew	862	0,09
8	sex == male	1731	class == crew	862	0,08
9	class == crew	885	age == adult && sex == male && survived == no	670	0,06
10	age == adult && survived == no	1438	class == crew && sex == male	670	0,05
11	survived == no	1490	class == crew && sex == male	670	0,04
12	survived == no	1490	class == 3rd && sex == male	422	0,03
13	class == 3rd	706	sex == female && survived == no	106	0,03
14	sex == female && survived == no	126	class == 3rd	106	0,03
15	class == 3rd && age == adult	627	survived == no	476	0,02
16	sex == female	470	class == 3rd	196	0,02
17	class == 2nd	285	sex == female	106	0,02
18	sex == female	470	class == 2nd	106	0,02
19	class == 3rd	706	age == child	79	0,02
20	age == child	109	class == 3rd	79	0,02
21	age == adult	2092	class == crew	885	0,02
22	age == adult	2092	sex == male && survived == no	1329	0,01
23	survived == yes	711	age == child	57	0,01

Tabelle 22: Weighted Relative Accuracy - minimumEvaluation = 0.005

#	Prämisse	p + n	Konklusion	p	Lift
1	class == 2nd && age == adult && survived == yes	94	sex == female	80	3,99
2	class == 2nd && survived == yes	118	sex == female	93	3,69
3	class == 1st && survived == yes	203	age == adult && sex == female	140	3,57
4	age == adult && survived == yes	654	class == 1st && sex == female	140	3,25
5	class == 3rd	706	age == child && sex == male && survived == no	35	3,12
6	age == child && survived == no	52	class == 3rd	52	3,12
7	sex == female && survived == no	126	class == 3rd	106	2,62
8	class == 3rd	706	sex == female && survived == no	106	2,62
9	sex == female && survived == no	126	class == 3rd && age == adult	89	2,48
10	class == 3rd && survived == yes	178	sex == female	90	2,37
11	age == child && sex == male	64	class == 3rd	48	2,34
12	age == adult && sex == male && survived == yes	338	class == crew	192	1,41
13	class == crew	885	age == adult && sex == male && survived == yes	192	1,41
14	class == 2nd && age == adult && sex == male	168	survived == no	154	1,35
15	class == 2nd && sex == male	179	age == adult && survived == no	154	1,32
16	age == adult && sex == male	1667	class == crew && survived == no	670	1,31
17	sex == male && survived == yes	367	class == crew	192	1,3
18	sex == female	470	class == 3rd	196	1,3
19	sex == male	1731	class == crew && survived == no	670	1,27
20	class == crew	885	age == adult && sex == male && survived == no	670	1,25
21	survived == no	1490	class == 3rd && age == adult && sex == male	387	1,24
22	sex == female	470	class == 3rd && age == adult	165	1,23
23	class == 3rd && sex == male	510	survived == no	422	1,22
24	age == adult && sex == male	1667	class == 2nd && survived == no	154	1,22
25	age == adult && sex == male	1667	class == crew && survived == yes	192	1,2
26	age == adult && survived == no	1438	class == crew && sex == male	670	1,19
27	sex == male	1731	age == adult && survived == no	1329	1,18
28	survived == no	1490	class == crew && sex == male	670	1,15
29	age == adult	2092	class == 2nd && survived == no	167	1,05
30	age == adult	2092	class == crew	885	1,05
31	class == 3rd	706	age == adult && survived == no	476	1,03
32	age == adult	2092	sex == male && survived == no	1329	1,03
33	age == adult	2092	sex == male && survived == yes	338	0,97
34	survived == yes	711	age == adult	654	0,97
35	age == adult	2092	class == 3rd && sex == male && survived == no	387	0,96
36	class == 2nd	285	age == adult	261	0,96
37	age == adult	2092	sex == female	425	0,95
38	sex == male	1731	class == 3rd && age == adult	462	0,94
39	survived == yes	711	class == crew	212	0,74

Tabelle 23: Lift - minimumEvaluation = 0.5

#	Prämisse	p + n	Konklusion	p	Correlation
1	age == adult && survived == no	1438	sex == male	1329	0,46
2	sex == male	1731	age == adult && survived == no	1329	0,46
3	survived == yes	711	sex == female	344	0,46
4	survived == no	1490	sex == male	1364	0,46
5	sex == female	470	survived == yes	344	0,46
6	age == adult && sex == male	1667	class == crew	862	0,41
7	class == crew	885	age == adult && sex == male	862	0,41
8	sex == male	1731	class == crew	862	0,38
9	class == 1st	325	age == adult && survived == yes	197	0,28
10	sex == female && survived == no	126	class == 3rd	106	0,27
11	class == 3rd	706	sex == female && survived == no	106	0,27
12	class == crew	885	age == adult && sex == male && survived == no	670	0,26
13	age == child && survived == no	52	class == 3rd	52	0,23
14	class == 3rd	706	age == child && survived == no	52	0,23
15	sex == female	470	class == 3rd && survived == yes	90	0,21
16	age == adult && survived == no	1438	class == crew && sex == male	670	0,21
17	age == child	109	class == 3rd	79	0,2
18	age == child	109	class == 2nd && sex == male && survived == yes	11	0,19
19	age == adult	2092	class == crew	885	0,19
20	survived == no	1490	class == 3rd && age == adult && sex == male	387	0,18
21	survived == no	1490	class == crew && sex == male	670	0,17
22	age == adult	2092	sex == male && survived == no	1329	0,14

Tabelle 24: Correlation - minimumEvaluation = 0.1