Karlsruhe Institute of Technology

Optimieren von POS-Tagger-Ergebnissen

Bachelorarbeit
von

Viktor Kiesel

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Ralf H. Reussner
Betreuender Mitarbeiter:	Dipl.-Inform. Sebastian Weigelt
Zweiter betr. Mitarbeiter:	Dipl.-Inform.Wirt Mathias Landhäußer

Bearbeitungszeit: 02.12.2015 – 01.04.2016

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 01. April 2016

.....

(**Viktor Kiesel**)

Kurzfassung

Ziel dieser Arbeit ist es, Wortartmarkiererergebnisse durch die Nutzung von Klassifikationsverfahren zu optimieren. Die Wortartmarkierung ist in der Computerlinguistik einer der ersten Verarbeitungsschritte. Um Folgefehler zu vermeiden ist es daher nötig, diese möglichst fehlerfrei durchzuführen. In dieser Arbeit wurde deshalb diese Problemstellung analysiert, eine softwaretechnische Lösung entworfen und implementiert. Der Entwurf sieht zwei separate Programme vor, die beide jeweils einen Teilschritt zur Lösung des Problems beitragen. Das erste Programm koordiniert neun Wortartmarkierer mit insgesamt elf Modellen, um Texte zu markieren. Die Ausgabe des ersten Programms wird vom zweiten Programm genutzt, um mit Hilfe von 14 Klassifikationsverfahren Modelle zu trainieren, Texte zu markieren und die Verfahren zu evaluieren.

Die Evaluation der Arbeit fand auf den Korpora WSJ, NLCI und PARSE statt. Um die Ergebnisse der Klassifikationsverfahren mit den Wortartmarkierern vergleichen zu können, wurde eine Performanzanalyse durchgeführt, um auf den Korpora die besten Markierer zu finden. Dabei wurde festgestellt, dass das gesamte Verfahren auf allen Korpora eine Verbesserung brachte. Neben der Frage welche Klassifikationsverfahren zu einer Verbesserung führen, wurde auch die benötigte Mindestgröße des Trainingskorpus untersucht. Insgesamt konnte auf dem WSJ-Korpus eine Verbesserung durch J48 auf 98,85% korrekt markierter Wörter vom besten Markierer Jitar mit 98,54% erreicht werden. Das Verfahren übertraf den Markierer bereits bei einer Trainingskorpusgröße von 9409 Wörtern. Auf dem NLCI-Korpus fand eine Verbesserung durch IBk auf 99,11% zum Stanford-Markierer mit 96,17% statt. Die Mindestgröße des Trainingskorpus für eine Verbesserung betrug 363 Wörter. Beim PARSE-Korpus wurde eine Verbesserung durch IBk mit 98,57% von 90,66% des Stanford-Markierers erreicht. Mit einem Trainingskorpus von 109 Wörtern konnte auf diesem Korpus eine Verbesserung erreicht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Das Thema dieser Arbeit ist die Optimierung von Wortartmarkiererergebnissen. Wortartmarkierer sind Programme aus dem Bereich der Computerlinguistik. Sie nutzen meist ein vorher trainiertes Modell, um den Wörtern eines Satzes die entsprechenden Wortarten zuzuweisen. Dies ist für viele Anwendungen in der Computerlinguistik einer der ersten Verarbeitungsschritte. Es ist daher wichtig, die Ergebnisse der Wortartmarkierung zu optimieren, um Folgefehler zu vermeiden.

Ein Beispiel für eine Anwendung, die Wortartmarkierer verwendet, ist das Generieren von Animationen aus einem Drehbuch. Der Lehrstuhl für Programmiersysteme des Karlsruher Instituts für Technologie arbeitet an dem Projekt „Programmieren in natürlicher Sprache“ mit dem Programm Alice[CAB+00] an einer solchen Anwendung. Im Rahmen dieses Projektes wurde der NLCI-Korpus[Ham12] erstellt. Er enthält erzählende Texte, aus denen das Programm Alice-Animationen erstellt. Die Markierung der Wörter des Satzes mit ihrer Wortart ist der erste Verarbeitungsschritt. Werden Wörter falsch klassifiziert, so entstehen Folgefehler, die dazu führen können, dass eine weitere Verarbeitung des Satzes nicht möglich ist. Dies kann zum Beispiel dazu führen, dass ein Akteur keine Handlung durchführt.

Die Wortartmarkierer werden in der Regel auf einem großen Korpus wie dem „Wallstreet Journal“-Korpus[MMS93] (WSJ-Korpus) trainiert. Der WSJ-Korpus besteht aus Zeitungsartikeln. Dies führt dazu, dass Texte, die sich mit anderen Themenbereichen beschäftigen, schlechter markiert werden. Aber selbst wenn ein Wortartmarkierer auf dem Korpus trai-

Tabelle 1.1.: Beispielsatz zur Optimierung der Wortartmarkiererergebnisse durch Mehrheitswahl

	Wortartmarkierer			Wortart
Wort	A	B	C	korrekt
The	DT	DT	DT	DT
white	NN	JJ	JJ	JJ
Bunny	NN	NN	JJ	NN
jumps	VBZ	VBZ	NN	VBZ
to	TO	TO	TO	TO
Alice	NNP	NN	NNP	NNP
.	.	.	.	.

niert wird, so führt das bei geringer Korpusgröße zu Problemen. Wird beispielsweise ein auf dem WSJ-Korpus trainierter Wortartmarkierer auf dem NLCI-Korpus evaluiert, so erreicht er eine Genauigkeit von 96,17%. Trainiert man ihn auf dem NLCI-Korpus, so kann er auf ebendiesem seine Genauigkeit auf 98,27% erhöhen. Für eine weitere Erhöhung der Genauigkeit müsste der Trainingskorpus vergrößert werden. Ist dies nicht möglich, dann ist eine weitere Erhöhung der Genauigkeit mit einem einzigen Wortartmarkierer nicht möglich. Durch Kombination mehrerer Wortartmarkierer aber könnte eine weitere Erhöhung erreicht werden.

In Tabelle 1.1 ist ein Satz aus dem NLCI-Korpus beispielhaft dargestellt. Drei Wortartmarkierer wurden verwendet, um diesen Satz zu markieren. Für die Wortartmarkierungen werden Abkürzungen verwendet. Vergleicht man die Ergebnisse der Wortartmarkierer mit der tatsächlichen Wortart, so stellt man fest, dass keiner der drei Wortartmarkierer den Satz alleine ganz korrekt markiert. Es entstehen also immer Folgefehler, sofern nur ein einziger Markierer genutzt wird. Kombiniert man aber die drei Markierer, indem man das Ergebnis auswählt, das die Mehrheit der Markierer ausgibt, so wird der Satz vollständig korrekt markiert. Das Ergebnis der Wortartmarkierer wurde optimiert, Folgefehler vermieden.

Der Mehrheitsentscheid ist davon abhängig, dass die Mehrheit aller genutzten Markierer das Wort korrekt markiert. Dies ist nicht garantiert, so erreicht zum Beispiel die einfache Mehrheitswahl mit elf Markierern auf dem NLCI-Korpus eine Genauigkeit von 97,65%. Dies ist zwar eine Verbesserung zu den genutzten Wortartmarkierern, aber kein optimales Ergebnis. Außerdem kann das Problem entstehen, dass Markierer, die in Spezialfällen richtig liegen, im Allgemeinen bewirken, dass ihr Einfluss zu falschen Ergebnissen führt. Durch das Hinzufügen weiterer Wortartmarkierer kann sich so das Gesamtergebnis verschlechtern. Viele der beschriebenen Probleme lassen sich durch komplexere Klassifikationsverfahren beseitigen. In der Arbeit von IBM (Research)[FBCC$^+$10] wurde gezeigt, dass durch ein Ensemble von Klassifikationsverfahren die Ergebnisse eines einzelnen Verfahrens übertroffen werden können. In dieser Arbeit soll daher überprüft werden, ob eine Optimierung durch Nutzung maschineller Lernverfahren auch bei Wortartmarkierern möglich ist. Weiterhin soll das Ausmaß der erreichten Verbesserung gegenüber dem besten Markierer untersucht werden.

Die Struktur dieser Arbeit folgt der logischen Aufbauweise zum Lösen des Problems. Nötige Grundlagen zum Verständnis der Arbeit werden in Kapitel 2 vermittelt. Zuerst werden einige Begriffe zur Funktionsweise von Wortartmarkierern definiert. Der Hauptteil des Kapitels beschäftigt sich dann mit Klassifikationsverfahren und ihrer Darstellung. Abschließend wird die Softwareumgebung Weka vorgestellt und die für die Evaluation nötigen Grundlagen werden erläutert.

Auf die Grundlagen folgt in Kapitel 3 ein Überblick über verwandte Arbeiten. Dabei werden zunächst Methoden vorgestellt, die eine dieser Arbeit ähnliche Idee verfolgen. Danach werden Arbeiten beschrieben, die Methoden im Bereich Sprachverarbeitung nutzen. Eine weitere Fokussierung auf das Thema findet statt, indem Arbeiten betrachtet werden, die Wortartmarkierer nutzen. Die meisten Verfahren beschäftigen sich mit englischsprachigen Eingaben, weswegen am Ende des Kapitels anderssprachige Arbeiten vorgestellt werden.

In Kapitel 4 wird die Problemstellung ausführlich definiert und analysiert. Auf der Analyse aufbauend werden die verwendeten Goldstandards und Wortartmarkierer beschrieben. Dabei wird neben dem NLCI-Korpus auch auf den PARSE-Korpus[WT15] genauer eingegangen, welcher aus mündlichen Anweisungen an einen Roboter besteht. Durch eine Performanzanalyse auf dem Goldstandard werden die Wortartmarkierer anhand ihrer Qualität sortiert. Die besten Wortartmarkierer dienen für die Evaluation als Vergleich. Aus der Analyse hervorgehend wird eine softwaretechnische Lösung entworfen. Dieser Entwurf

besteht aus zwei von einander unabhängigen Programmen, die ausführlich beschrieben werden.

Probleme und Herausforderungen, die sich beim Implementieren der softwaretechnischen Lösung stellten, werden in Kapitel 5 aufgezeigt und die gewählten Lösungen vorgestellt. Weiterhin wird auf Implementierungsdetails eingegangen.

Die Programme beinhaltet die einfache Möglichkeit das Verfahren zu evaluieren. Neben der zentralen Hypothese dieser Arbeit, dass die Kombination mehrerer Wortartmarkierer durch Klassifikationsverfahren zu einer Verbesserung führt, werden in Kapitel 6 zur Evaluation weitere Hypothesen aufgestellt und mit Hilfe von zwei Kreuzvalidierungsverfahren überprüft. Mit der 10-fachen Kreuzvalidierung wird festgestellt, ob eine Verbesserung durch das gesamte Verfahren stattfand. Die Monte-Carlo-Kreuzvalidierung erlaubt die Findung der benötigten Mindestkorpusgröße zum ergebnisverbessernden Training der Klassifikationsverfahren. Da nicht bekannt ist, welche Klassifikationsverfahren Erfolg versprechen, werden verschiedene Verfahren getestet und verglichen.

Abschließend fasst Kapitel 7 die Arbeit zusammen und liefert darauf aufbauende mögliche Ansätze und Ideen zur weiteren Forschung.

2. Grundlagen

Viele Anwendungen in der Computerlinguistik sind von einer möglichst optimalen Wortartmarkierung abhängig. Diese Aufgabe wird von Wortartmarkierern meist nicht optimal erledigt. Die Ergebnisse einer Wortartmarkierung lassen sich daher, durch Verwendung mehrerer Wortartmarkierer und einer einfachen Mehrheitswahl zwischen den Ergebnissen, optimieren. Während einfache Auswahlmethoden, wie der Mehrheitsentscheid eine Verbesserung der Genauigkeit bringen können, sollen in dieser Arbeit auch komplexere Auswahlmethoden überprüft werden. Deshalb ist einiges an Wissen über das maschinelle Lernen notwendig. Der Fachbereich des maschinellen Lernens ist in der Informatik von großer Bedeutung. Das Ziel ist es, neue Informationen aus bereits bekanntem Wissen zu gewinnen. Ein Algorithmus soll hierbei nicht einfach nur einen Datensatz aus einer Datenmenge abrufen, sondern aus der Datenmenge Regelmäßigkeiten und Muster erlernen. Durch Verwendung dieser Muster sollen neue vorher unbekannte Daten überprüft und in entsprechende Gruppen bzw. Klassen eingeteilt werden. Die Ausgabe der einzelnen Wortartmarkierer soll hierbei als Eingabe für die Klassifikationsalgorithmen dienen.

Dem logischen Ablauf folgend, werden zuerst für das Verständnis der Funktionsweise der genutzten Wortartmarkierer in Abschnitt 2.1 einige Fachbegriffe definiert. In Abschnitt 2.4 dieses Kapitels wird dann ein Überblick über die generelle Klassifikation gegeben. Zum Abschluss wird in Abschnitt 2.3 die Softwareumgebung Weka kurz vorgestellt und die Evaluationsgrundlagen in Abschnitt 2.4 erläutert.

2.1. Grundlagen der Wortartmarkierung

Ein Teilbereich des maschinellen Lernens ist die Wortartmarkierung. Die Kenntnis der Funktionsweise generellerer Klassifikationsalgorithmen hilft daher beim Verständnis der Funktionsweise der Wortartmarkierer. Klassifikationsalgorithmen, die Wörtern eines Satz ihren entsprechende Wortartmarkierungen, mit Hilfe eines vorher trainiertem Models, zuweisen, nennt man Wortartmarkierer (engl. *„Part of Speech“*-Tagger). Die Qualität dieser Markierer ist abhängig von der Qualität des Models.

Wie bei jeder Wissenschaft gibt es auch im Bereich der Wortartmarkierungen Begriffe, die wiederholt vorkommen und nicht sofort ersichtlich sind. Dieses Unterkapitel soll diese fachsprachlichen Ausdrücke erläutern.

2.1.1. Grundbegriffe

Ein **Token** ist eine Zeichenkette, die von einem Wortartmarkierer mit einer einzigen Markierung versehen wird. Ein Tokenisierer kümmert sich insbesondere darum, dass Namen wie „New York“ als ein einziges Token erkannt werden und dadurch korrekt als „New York/NNP“ markiert werden. Dies ist ein Vorarbeitsschritt, der vor der eigentlichen Wortartmarkierung stattfindet. Oft implementieren Wortartmarkierer eigene Tokenisierer.

Das **Lemma** eines Wortes ist die Grundform des Wortes ohne Kapitalisierung oder andere Modifikationen.

Ein **Merkmal** ist eine dem Token zugehörige Eigenschaft, die benutzt werden kann, um die Menge der möglichen Markierungen einzugrenzen oder die Wahrscheinlichkeitsverteilung der Markierungen zu verschieben. Großschreibung und Leerzeichen im Token dienen beispielsweise im Englischen dazu Eigennamen zu identifizieren.

Lexikalische Informationen sind alle jene Informationen, die sich rein aus dem Wort selber herausfiltrieren oder in einem Lexikon speichern lassen. Dies beinhaltet alle Merkmale des Wortes und die Auftrittswahrscheinlichkeit des Wortes, welche aus einem Trainingskorpus ermittelt wird.

Die **Genauigkeit** ist eine Metrik, die beschreibt, wie viele Wörter auf einem unbekannten Text durchschnittlich korrekt markiert werden. Alternative Metriken wären die Prozentzahlen an komplett korrekt markierten Sätzen und die Genauigkeit der unbekannten, im Trainingsdatensatz nicht vorkommenden, Wörter. Auch können diese Werte nach Korpora getrennt angegeben sein.

Der heutige Stand der Technik entspricht einer Genauigkeit von ungefähr 97,3% auf dem „*Wall Street Journal*“-Korpus trainiert und ebendort getestet[Man11].

Die **Konfidenz** gibt an wie sich ein Wortartmarkerer ist, dass ein Wort mit der korrekten Markierung versehen wurde.

Übergangswahrscheinlichkeit ist die Auftrittswahrscheinlichkeit, die eine bestimme Sequenz von Wortarten besitzt, unabhängig von den konkreten Wörtern.

Überanpassung entsteht wenn das Modell eines Klassifikationsverfahren zu stark an die Trainingsdaten angepasst ist. Dies führt zu einer niedrigeren Genauigkeit des Modells auf allgemeinen Daten.

Glättung ist ein Rauschunterdrückungsverfahren. Ziel dabei ist es Modelle, die Ausreißern auf Grund von zu wenig Trainingsdaten oder Überanpassung hohen Einfluss geben in ein generelleres Modell umzuwandeln. Es gibt viele Methoden, die diesen Zweck erfüllen.

N-Gramme bezeichnen eine Zerlegung eines Wortes oder Wortgruppen in alle möglichen N-Zeichen langen Zeichen-, bzw Wortketten, ohne dass die Reihenfolge der Zeichen geändert wird. Sie werden für gewöhnlich dazu benutzt die Übergangswahrscheinlichkeiten der Wortartsequenzen zu berechnen.

Als **Fenster** wird bei einem großen Datensatz, der nicht direkt komplett verarbeitet werden kann, der aktuell zu bearbeitende Ausschnitt bezeichnet. Dies kann beispielsweise bei der Wortartmarkierung das aktuelle Wort und die beiden Nachbarn sein.

Bei einer **globalen Verarbeitungsmethode** wird der gesamte Kontext gleichzeitig verarbeitet. Alle Merkmale haben die Möglichkeit jeden anderen Teilkontext zu beeinflussen und sich von diesem beeinflussen zu lassen.

Bei einer **lokalen Verarbeitungsmethode** wird ein Fenster genutzt. Merkmale können daher nur den lokalen, aktuell im Fenster bearbeiteten Kontext beeinflussen, wenn sie sich innerhalb des Fensters befinden.

2.1.2. Markow-Modelle

Verborgene-Markow-Modelle (engl. Hidden Markov Models (HMM)) sind die Modellierung eines Systems mit Hilfe einer Markow-Kette, bei welchem die Zustände unbekannt sind. Markow-Ketten bestehen aus Zuständen und stochastisch Zustandsübergängen. Die Wahrscheinlichkeit für eine Zustandssequenz kann durch Ablaufen der Zustände und die Multiplikation der entsprechenden Zustandsübergangswahrscheinlichkeiten ermittelt werden. Bei HMM sind die eigentlichen Zustände hingegen verborgen und nur die von diesen Zuständen ebenfalls stochastisch abhängigen Emissionen sind offen bekannt. Da eine Emission von mehreren Zuständen stammen kann, gilt es die für diese Emissionssequenz wahrscheinlichste Zustandssequenz zu berechnen.

Maximum-Entropie-Markow-Modelle berechnen die Wahrscheinlich, dass ein Wort zu einer bestimmten Wortart gehört, indem die Existenz manuell definierter Merkmale, anstatt reiner Frequenzanalyse, überprüft werden. Wie auch bei HMMs werden die Übergangswahrscheinlichkeiten unidirektional, in einem Fenster, berechnet.

Der **Viterby-Algorithmus** wird dafür verwendet, um bei einem HMM die wahrscheinlichste Sequenz der Zustände, abhängig von einer gegeben Sequenz an Emissionen, zu berechnen. Er kann überall angewendet werden, wo Muster erkannt werden sollen.

2.1.3. Grundbegriffe der Syntaxanalyse

Neben den Wortartmarkierern, die auf einer sequentiellen Basis arbeiten, existieren auch Syntaxanalysierer, die einen Baum aufbauen, der die grammatikalische Struktur des Satzes widerspiegelt. Anhand der Blätter und Knoten dieses Baumes lassen sich die einzelnen Wortarten ablesen. Um die Bäume zu generieren, werden meist aus den Wörtern Hypothesen, das heißt potentielle Kandidaten für eine Markierung, gewonnen. Bei bekannten Wörtern kann dies durch eine Wortfrequenzanalyse geschehen. Bei unbekannten Wörtern werden aus der Umgebung des Wortes mögliche Kandidaten für die Wortarten gewonnen. Sind die Grammatiken und Hypothesen bekannt, so ist das Finden des passenden Baumes eine Suche im Raum aller möglichen Bäume, welche oft, auf Grund der hohen Komplexität, mit einer heuristischen Methode, der *Strahlensuche*, erledigt wird.

Eine **Hypothese** ist die Menge von möglichen Markierungen, die für ein Token in Frage kommen könnten. Durch Weiterverarbeitung dieser Hypothese wird die endgültige Markierung gefunden. Sie ist vor allem beim Parsen hilfreich, um den Suchraum einzuschränken und somit die Suchzeit zu verringern.

Ein **Dependenzbaum** ist der Ableitungsbaum einer **Dependenzgrammatik**. Dieser Baum stellt die Abhängigkeitsbeziehung zwischen den Wörtern in einem Satz dar.

Die **Strahlensuche** ist eine spezielle Form der Breitensuche, bei der nur eine vorher fest definierte Anzahl an vielversprechendsten Ästen weiter durchsucht wird. Da Äste verworfen werden, kann die Strahlensuche nicht garantieren, dass das Endergebnis das bestmöglichste Resultat ist.

2.2. Grundlagen der Klassifikation

In Anwendungsbereich des maschinellen Lernens werden Klassifikationsverfahren dazu genutzt, um Daten zu klassifizieren. Die Klassifikationsverfahren nutzen hierfür trainierte Modelle. In dieser Arbeit werden verschiedene Klassifikationsverfahren dazu genutzt, um Wortartmarkierer zu kombinieren. Da nicht Verfahren, für den geplanten Anwendungszweck sinnvoll anwendbar sind, beschränkt sich diese Ausarbeitung nur auf Verfahren, bei denen die Klassen bereits vor dem Training bekannt sind. Auch werden Algorithmen

Tabelle 2.1.: Wetterdatensatzbeispiel

Eingabedaten			Klassifikation
Nebel	Temperatur	Regen	Schnee
ja	kalt	nein	nein
ja	heiß	ja	nein
nein	kalt	ja	ja
nein	heiß	nein	nein
ja	kalt	ja	ja

und Modifikationen, die explizit mit numerischen Werten arbeiten, ignoriert, da nur ein abgeschlossener Attributsatz existiert.

Als letzter Punkt sei erwähnt, dass dies nur einen Überblick über die verwendeten Verfahren und Algorithmen geben soll und viele Details nicht beschrieben wurden.

Als Grundlage für dieses Kapitel diente das Buch „Data Mining“[WFH11] von Witten et al.

2.2.1. Darstellungsformen der Klassifikation

Ein Klassifikationsalgorithmus nutzt für gewöhnlich eine möglichst große Menge an verschiedenster Daten und verwendet diese, um ein Modell zu trainieren.

Ein Datum aus einer Datenmenge wird im Bereich des maschinellen Lernens als Instanz bezeichnet. Eine Instanz ist durch ihre Attribute und die zu diesen Attributen gehörigen Werte definiert. Der Werteraum eines Attributes ist abhängig vom Datentyp des Attributes, welcher sich in drei Klassen einteilen lassen:

1. **Klasse der geschlossenen, abzählbaren Attribute:** Der Werteraum dieser Klasse können eine endliche Anzahl an Werten annehmen. Als Beispiel wäre hier der Wochentag zu nennen.
2. **Klasse der numerischen Attribute:** Der Werteraum dieser Klasse sind reelle Zahlen und im Allgemeinen ohne Begrenzung.
3. **Klasse der Strings:** Diese Attribute bestehen aus einer Reihe zusammengesetzter geordneter endlicher Klassen. Ein Beispiel für diese Klasse sind Wörter.

Das Eingabeformat besteht aus der Deklaration der Attributtypen und den eigentlichen Dateninstanzsätzen mit jeweils konkreten Werten. Mehrere Instanzen werden tabellarisch dargestellt.

Werden die Instanzen verarbeitet, so gibt es mehrere Möglichkeiten die Trainingsinstanzen, die neuen Instanzen und aus den Trainingsdaten gewonnene Informationen darzustellen. Diese werden hier besprochen.

2.2.1.1. Tabellen

Die Trainingsdaten werden für gewöhnlich in Form einer Liste oder Tabelle übergeben. Es liegt daher nahe die Ausgangsdaten in der gleichen tabellarischen Form darzustellen. Alle Instanzen werden dabei, unabhängig ihrer Eigenschaften, gleich dargestellt. Dies kann bei großen Datenmengen sehr schnell unübersichtlich werden. In Tabelle 2.1 und Abschnitt A ist ein Wetterdatensatz in dieser Form dargestellt. Die erste Zeile enthält die Attribute unter denen die Werte der einzelnen Instanzen stehen. Die rechte Spalte zeigt die Klasse zu der die Instanz gehört.

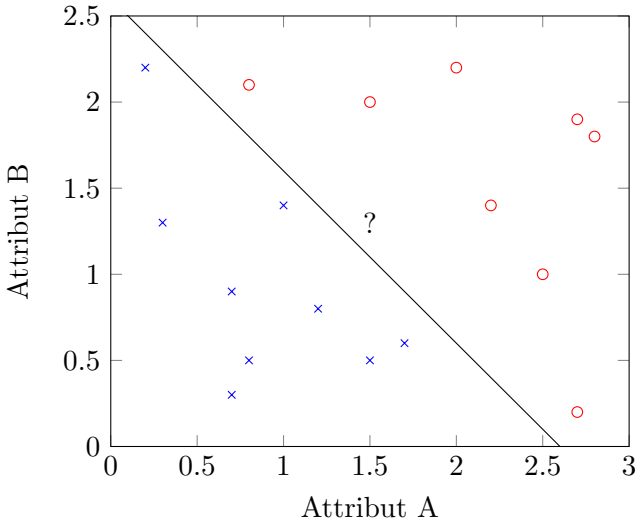

Abbildung 2.1.: Beispiel für die Klassifikation auf linearen Modellen. Die Klasse von Instanzen ist abhängig von ihrer Lage zur Gerade. `?` stellt hierbei eine unbekannte, noch zu klassifizierende Instanz dar.

2.2.1.2. Lineare Modelle

Lineare Modelle sind anders als Tabellen eine sehr übersichtliche Darstellungsweise, wenn die Klasse von weniger als drei Attributen abhängt. Hierbei wird der Werteraum der einzelnen Attribute auf entsprechende Achsen in einem kartesischen Koordinatensystem abgebildet. Dies ermöglicht eine mathematische/algebraische Betrachtung der Klassifikation und die Nutzung von Regressionsalgorithmen aus der Statistik. Die *Methode der kleinsten Quadrate* wird hierfür oft angewendet. Ähnlich kann auch eine Klassifikation stattfinden, indem der Raum durch eine lineare Funktion, abhängig von einer Konstanten und den gewichteten Attributwerten getrennt wird.

In Abbildung 2.1 wird ein Datensatz mit zwei numerischen Attributen durch eine Gerade in zwei Klassen eingeteilt. Die Position einer Instanz wird durch deren Attributwerte definiert. Instanzen links, bzw unter der Geraden, gehören hierbei der Klasse `X` an. Instanzen, die durch ihre Attribute eine Position rechts, bzw. oberhalb der Gerade zugewiesen bekommen, werden als Klasse `O` klassifiziert. Die unklassifizierte Instanz `?` wird dadurch der Klasse `O` zugeteilt.

2.2.1.3. Entscheidungsbäume

Die Entscheidungsbäume stellen die Informationen, die aus den Trainingsdaten gewonnenen wurden, in Form eines Baumes dar. Dabei wird durch mehrere aufeinanderfolgende Entscheidungsknoten ein Entscheidungsbaum aufgebaut, bei dem die Blätter auf Klassen zeigen. Jeder Knoten testet die Menge auf eine Eigenschaft der Attribute und teilt die Menge der Instanzen, die diesen Knoten erreicht haben, in entsprechende Untermengen auf. Alle Instanzen, die einen Unterknoten erreichen, haben dadurch ähnliche Werte. Der Baum wird von der Wurzel aus durchlaufen, bis ein Blatt erreicht wird. Die Instanz wird bei Erreichen eines Blattes klassifiziert.

In Abbildung 2.2 ist ein einfacher Entscheidungsbaum dargestellt. Dieser ordnet jeder Instanz einen Wert der binären Klasse `Schnee` zu. Die Wurzel des Baumes ist das Attribut `Temperatur`.

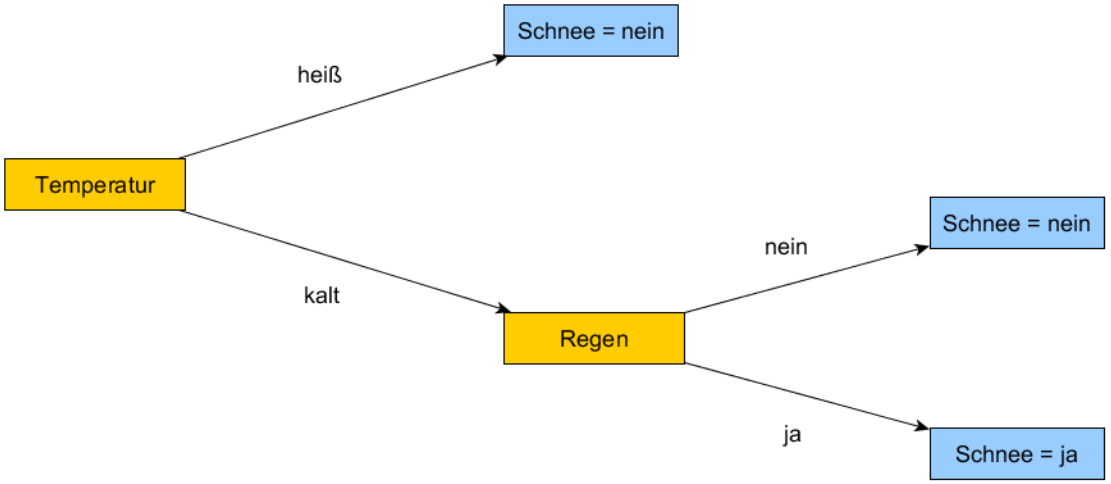

Abbildung 2.2.: Entscheidungsbaumbeispiel

2.2.1.4. Regeln

Die Regeln sind mit den Entscheidungsbäumen verwandt. Deshalb lassen sich die beiden Konzepte oft in einander übersetzen. Sie sind anders als Entscheidungsbäume hingegen kompakter, da nicht der ganze Baum durchlaufen werden muss, sondern einzelne Regeln direkt die Klasse spezifizieren.

Trifft eine Regel zu, so wird die Instanz der entsprechenden Klasse zugeordnet. Durch die Modularität der Regeln sind diese leicht durch weitere Regeln erweiterbar. Um zu spezifische Regeln zu vermeiden, können *Konfidenz-* und *Abdeckungswerte* verwendet werden. Eine Regel wird als gültig akzeptiert, wenn eine bestimmte Mindestanzahl an Instanzen diese Regeln erfüllt. Auch ist es möglich Regelausnahmen hinzuzufügen, bei denen diese Regel nicht gelten soll. Dadurch ist es möglich ist einen komplexen Baum aus einzelnen Regeln und Regelausnahmen zu definieren, bei dem es zu mehreren verschachtelten Ausnahmen kommt.

Zwei Regelsätze, die eine Instanz der binären Klasse `Schnee` zuordnen, sind in Unterunterabschnitt 2.2.1.4 dargestellt. Beide Regelsätze sind äquivalent, unterscheiden sich aber dadurch, dass Regelsatz 2 eine Ausnahme verwendet, um eine zu breit abgedeckte Menge an Instanzen zu verkleinern. Regelsatz 1 spezifiziert die kleinere korrekte Menge direkt.

Regelsatzbeispiel

```
---Regelsatz 1:---
IF Regen = ja AND Temperatur = kalt THEN Schnee = ja
Default Schnee = Nein

---Regelsatz 2:---
IF Regen = ja THEN Schnee = ja
    EXCEPT IF Temperatur = heiß
Default Schnee = Nein
```

2.2.1.5. Instanzbasierte Repräsentationen

Die Grundidee der Instanzbasierten Repräsentation ist es, bei der Klassifikation von neuen Instanzen, die bereits bekannten und klassifizierten Instanzen zu durchsuchen und dabei

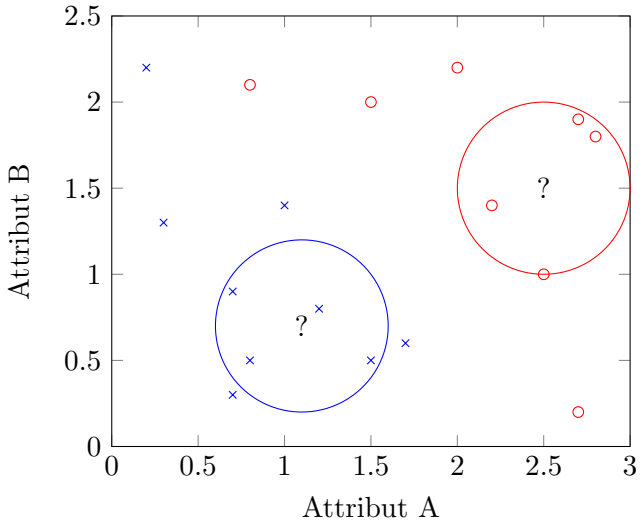

Abbildung 2.3.: Beispiel für die Klassifikation von Instanzen mit zwei Attributen durch ein nächste-Nachbarn-Verfahren. ? stellt hierbei unbekannte, noch zu klassifizierende Instanzen dar.

die Instanz auszuwählen, die der unbekannten Instanz am nächsten ist. Im Vergleich zu anderen, findet der Hauptteil der Arbeit also beim Klassifizieren der neuen Instanzen statt und nicht beim Trainieren.

Ein Beispiel für einen instanzbasierten Algorithmus ist die *k-nächste-Nachbarn-Klassifikation (kNN)*. Hierbei werden die k nächstgelegenen Nachbarn der Instanz betrachtet und die Instanz mit der am meisten unter den Nachbarn vorkommenden Klasse markiert. Als Distanz kann die gewichtete euklidische Distanz verwendet werden, wobei bei Attributen der geschlossenen Attributklasse die Distanz 1 beträgt, wenn die Werte unterschiedlich sind, sonst 0. Um die Laufzeit zu verbessern, werden für gewöhnlich nur wichtige, das heißt nur Instanzen, die nötig zum Klassifizieren sind, gespeichert.

Als Alternative zur kNN-Methode kann auch eine Region markiert werden, und die Instanz wird anhand der Region in der sie liegt klassifiziert. Liegt die Instanz in keiner Region, so wird die am nächsten liegende Region ausgewählt.

Ein Beispiel für eine Klassifikation mit Hilfe der Instanzen der näheren Umgebung ist in Abbildung 2.3 zu sehen. Hierbei wird den unbekannten Instanzen ? die Klasse zugewiesen, die in der Umgebung am öftesten vorkommt.

2.2.2. Klassifikationsalgorithmen

Während im vorherigen Kapitel die verschiedenen Darstellungsformen der Daten und die aus den Trainingsdatensatz gewonnen Informationen beschrieben wurden, geht es in diesem Kapitel um die konkreten Algorithmen. Zur Einführung eines jeden Algorithmus wird jeweils eine einfache Grundvariante vorgestellt und dann gegebenenfalls mehrere komplexere Erweiterungen.

2.2.2.1. Einfache Regeln inferieren

Zu Beginn soll einer der einfachsten Möglichkeiten vorgestellt werden, die *1-Regel (1R)* aus einem Trainingssatz zu gewinnen. Die Idee hinter dem 1R-Algorithmus ist es, einen Entscheidungsbaum aufzubauen, der der nur aus der Wurzel besteht.

Der Entscheidungsbaum, welchen der 1R-Algorithmus aufbaut, arbeitet in zwei Schritten. Zuerst wird für jeden Wert jedes Attributes eine einzige Regel inferiert. Dies wird erreicht,

Tabelle 2.2.: Beispiel für 1R

	Attribut	Regelsatz	Fehlerrate	Gesamtfehlerrate
1	`Regen`	`ja` -> *`Schnee`* = `ja`	6/18	11/30
		`nein` -> Schnee = `nein`	5/12	
2	`Temperatur`	`heiß` -> *`Schnee`* = `nein`	4/10	13/30
		`kalt` -> *`Schnee`* = `nein`	9/20	
3	`Nebel`	`ja` -> *`Schnee`* = `nein`	7/17	13/30
		`nein` -> *`Schnee`* = `nein`	6/13	

indem für jeden Attributwert gezählt wird, welches die meist vorkommende Klasse ist. Als Regel für diesen Wert des Attributes wird die Abbildung auf diese meist vorkommende Klasse festgelegt. Dies wird für jeden Wert eines Attributes wiederholt und für jedes Attribut entsteht dadurch ein Regelsatz, der einen Wert auf die meist vorkommende Klasse abbildet.

Nachdem für jedes Attribut eine Menge von Regeln erstellt wird, wird der Regelsatz des Attributes mit der geringsten Fehlerrate ausgewählt, um die Klassifikation durchzuführen.

Tabelle 2.2 zeigt die fertig bearbeiten Information des Algorithmus. Als Eingabedaten galten mehrere Instanzen mit den Attributen `Regen`, `Temperatur` und `Nebel`. Klassifiziert wird auf die binäre Klasse *`Schnee`*, die entweder `ja` oder `nein` sein kann. Der vollständige Beispielsdatensatz ist im Anhang als Abschnitt A zu finden.

Es wurde für jedes Attribut ein Regelsatz abgeleitet, welcher abhängig vom Wert des Attributes die Instanz einer Klasse zuweist. Die Anzahl der Fehler auf dem Trainingsdatensatz einer jeden Regeln wird gezählt und die beste Regel ausgewählt. Die Fehlerrate des Attributs ergibt sich aus der Anzahl, der für diesen Attributwert falsch abgebildeten Instanzen, geteilt durch die Anzahl dieses Attributwertes. Die Summe der Anzahl aller möglichen Werte die das Attribut annehmen kann entspricht der Anzahl an Instanzen im Trainingsdatensatz. Die Gesamtfehlerrate eines Regelsatzes ist die Anzahl aller Fehler, die durch Nutzung des Regelsatzes entstehen würden geteilt durch die Anzahl der Instanzen.

Da Regelsatz 1 (`Regen`) die geringste Gesamtfehlerrate aufweist, würde dieser zur Klassifikation der unbekannten Instanzen von 1R ausgewählt werden.

Das Hauptproblem bei 1R ist, dass der Algorithmus Attribute, die viele einzigartige Werte haben, stark bevorzugt. Würde man ihn auf einen Datensatz anwenden, bei dem ein Attribut eine einzigartige Identifikationsnummer ist, so würde er jeder Nummer die Klasse der Instanz der Identifikationsnummer zuweisen. Da jede Nummer nur ein einziges mal vorkommt, werden so keine Fehler gefunden und die Klassifikation nach der Identifikationsnummer wird als bester Regelsatz ausgewählt. Offensichtlich wird diese Regelmenge bei der Klassifikation einer neuen Instanz mit unbekannter Identifikationsnummer versagen. Um dieses Problem der Überanpassung zu beseitigen, wird für gewöhnlich gefordert, dass die Anzahl der Werte eine Mindestanzahl überschreiten muss, bevor sie beachtet wird. Dazu werden wenn nötig mehrere Werte zu einem neuen Wert zusammenfasst.

Eine Erweiterung des 1R Algorithmus ist der *Hyperpipes*-Algorithmus. Hierbei wird statt einer einzigen Regel für alle Klassen, eine Regel für jede Klasse erstellt.

Anwendungsgebiet: Der 1R-Algorithmus wird oft als Komponente in einem Zusammenschluss mehrerer Klassifikatoren zum *Verstärken* genutzt. Die Viola-Jones-Methode[VJ01] nutzt diese, unter anderen, 1R-Komponenten, um Gesichter zu erkennen.

2.2.2.2. Statistische Modelle

Während das 1R-Verfahren aus den Trainingsdaten direkt Regeln lernt, sind die statistischen Methoden dazu gedacht Wahrscheinlichkeiten für die verschiedenen Klassen zu berechnen. Eine Instanz wird dabei als die Klasse klassifiziert bei der sie die höchste Wahrscheinlichkeit hat.

Die Wahrscheinlichkeit, dass der *Bayes-Klassifikator* eine Instanz auf eine bestimmte Klasse abbildet, ist in der Gleichung 2.1 dargestellt. Es berechnet sich aus dem Produkt der Einzelwahrscheinlichkeiten aller Instanzattribute und der Klassenverteilung. Die Instanz wird der Klasse mit der höhsten Wahrscheinlichkeit zugeordnet. Dabei wird angenommen, dass die Attribute von einander komplett unabhängig sind.

$$P\left(Instanz|Klasse\right) = \prod_{i=1}^{AnzahlAttribute} \Big(P\left(Attribut_i \text{ von } Instanz|Klasse\right)\Big) * P\left(Klasse\right) \tag{2.1}$$

Während diese naive Unabhängigkeitsannahme der Attribute zwar ein Ergebnis liefert, gibt es einige Schwierigkeiten. So führt ein Attributwert, welcher nicht in der Menge der Trainingsinstanzen vorkommt dazu, dass die Wahrscheinlichkeit der Instanz in der Klasse zu sein Null ist. Dieser Einspruch lässt sich dadurch entfernen, indem zu jedem Wert ein $\epsilon > 0$ addiert wird. Dadurch führt selbst das Fehlen eines Attributwertes zu einer Wahrscheinlichkeit größer Null.

Ein weiteres Problem der Unabhängigkeitsannahme ist, dass die Klasse der Instanz in der Realität unterschiedlich stark von den einzelnen Attributen abhängt. Dies lässt sich durch einfache Gewichtung der Attribute beseitigen.

Anwendungsgebiet: Der Bayes-Klassifikator kann trotz seiner Einfachheit bei vielen Klassifikationsproblemen verwendet werden. Ein beliebtes Anwendungsfeld ist die Dokumentenklassifikation[APK+00]. Dies kann die Einschätzung der Gemütslage eines Textes bis hin zur Entdeckung von unerwünschter Werbung in elektronischer Post sein.

2.2.2.3. Bayessche Netze

Die Bayesschen Netzte bestehen aus einem direkten azyklischen Graphen, bei dem jeder Knoten einem Attribut der Instanz entspricht. Jeder Knoten besitzt eine Wahrscheinlichkeitsverteilungstabelle, die abhängig von den Eingangsknoten ist. Die Kanten zwischen den Knoten zeigen damit die Abhängigkeiten zwischen den Attributen an. Das Produkt dieser Knoten ist eine *multivariate Verteilung*.

In Abbildung 2.4 ist ein einfaches Bayessches Netz dargestellt. `Schnee`, und damit die Wahrscheinlichkeitsverteilungstabelle, hängt hierbei von den beiden Attributen `Temperatur` und `Regen` ab. Da die Attribute `Temperatur` und `Regen` keine Eingangsknoten haben, ist die Wahrscheinlichkeit der Werte unabhängig von allen anderen Attributen.

Um ein Bayessches Netz zu trainieren werden zwei Komponenten benötigt. Die erste Komponente ist eine Funktion, um ein auf Daten basierendes Netz zu evaluieren. Als zweite Komponente wird eine Methode benötigt, die es ermöglicht den Raum der möglichen Netze durchsuchen zu können.

Sind die Kanten bereits bekannt, so ist das berechnen der Knotentabellen trivial. Es wird die relative Verteilung der assozierten Kombination im Trainingsdatensatz berechnet. Sind die Kanten hingegen unbekannt, so muss die Struktur durch eine Suche durch alle möglichen Netzstrukturen abgeschätzt werden. Dabei existiert das Problem der Überanpassung,

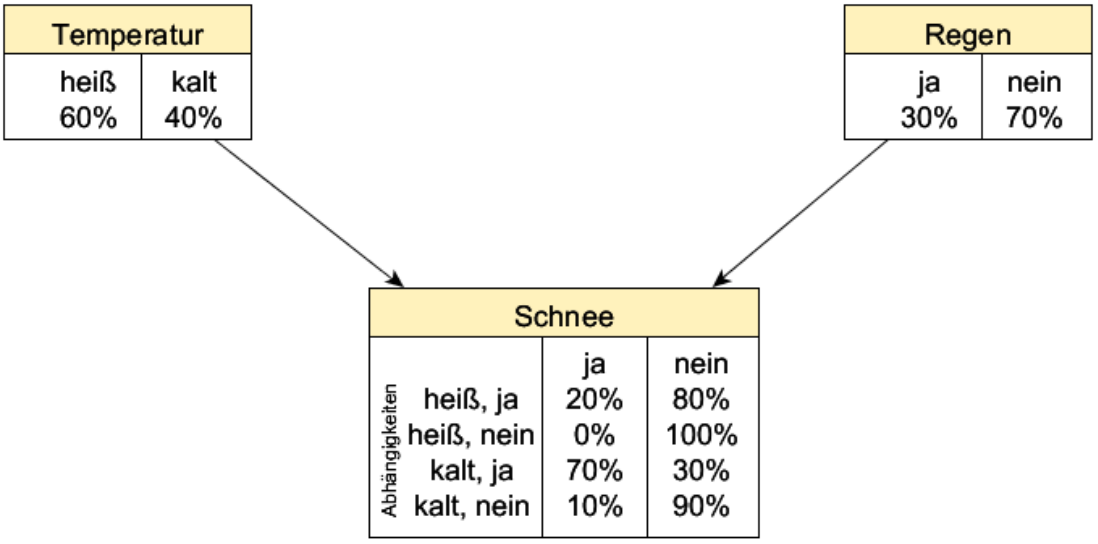

Abbildung 2.4.: Bayessches Netz für Stimmung

da das Hinzufügen weiterer Kanten auf dem gleichen Trainingsset immer vorteilhafter ist. Deshalb wird oft noch bei der Abschätzung ein Abzug für die Komplexität der Struktur des Netzwerkes einberechnet.

Die Knoten sind immer bekannt, da diese sich einfach durch Abbilden jedes Attributes auf einen neuen Knoten, erzeugen lassen.

Der *K2-Algoritmus* ist ein Lernalgorithmus für Bayessche Netze. Der Algorithmus startet mit einer geordneten Menge an Attributknoten und geht diese der Reihe nach durch. Dabei verbindet er bereits verarbeitete Knoten an den aktuell betrachteten Knoten an. Ziel hierbei ist es für den aktuellen Knoten die beste Kante zu finden. Sobald dies nicht mehr möglich ist, wird zum nächsten Knoten übergegangen. Da die ursprüngliche Anordnung der Attributknoten eine Rolle spielt, sollte der Algorithmus mehrmals mit einer anderen Sortierung der Knoten wiederholt werden.

Anwendungsgebiet: Die Bayesschen Netze werden oft als Expertensystem in Bereichen wie der Bioinformatik und Medizin eingesetzt[ZC05]. Hierbei soll das Programm aus einer Wissenbasis eine Handlungsempfehlung ableiten. Auch bei der Schlussfolgerung durch eine künstliche Intelligenz[Coo90] werden sie verwendet.

2.2.2.4. Entscheidungsbäume

Die Entscheidungsbäume klassifizieren die Instanzen indem diese die Attributtests der einzelnen Knoten von der Wurzel bis zu einem Blatt, welches ihnen die passende Klasse zuweist, durchlaufen.

Die Tiefe des Baumes hat hierbei einen Einfluss darauf wie schnell eine Instanz klassifiziert wird. Es ist vorteilhaft die Reihenfolge der Attribute die getestet werden möglichst so zu wählen, dass eine Instanz nur wenige Stufen passieren muss, um ein klassifizierendes Blatt zu erreichen. Dies wird dadurch ermöglicht indem die *Reinheit der Unterknoten*, und damit der Untermenge an Instanzen die diesen Unterknoten erreichen, berechnet wird. Nach der Berechnung, wird das Attribut mit der größten Differenz zur aktuellen Stufe ausgewählt. Ist ein Unterbaum rein, das heißt alle Instanzen sind der selben Klasse zugehörig, so wird an dieser Stelle statt eines Knotens ein Blatt mit der passenden Klasse angehängt. Damit wird erreicht das einzelne Unterbäume bereits nach wenigen Stufen in Blättern enden und damit klein bleiben.

Wie auch bei 1R können Attribute mit vielen selten vorkommenden Werten ein Problem darstellen, da die Reinheit ihrer Unterknoten sehr hoch ist. Um dies zu vermeiden kann man die Menge der Unterknoten mit einrechnen. Für gewöhnlich werden aber solche Attribute bereits im Vorfeld aussortiert.

Während der einfache Algorithmus zum Aufbau eines Baumes bereits für viele Tätigkeiten ausreichend gut funktioniert, so gibt es trotzdem noch Probleme damit, dass die so erzeugten Bäume viel zu spezifisch auf den Trainingsdatensatz angepasst sind. Der *C4.5-Algorithmus* ist eine Erweiterung des Standardalgorithmus zum Aufbau eines Entscheidungsbaumes. Er kann mit numerischen und fehlenden Werten umgehen und bekämpft die Überanpassung durch Beschneidung, beziehungsweise Zurechtstutzen der generierten Entscheidungsbäume. Bäume die zurechtgestutzt wurden, teilen Instanzen, die nicht im Trainingssatz enthalten sind, weniger spezifisch in Klassen auf. Dies erhöht die Genauigkeit auf einem unabhängigen Testdatensatz.

Zurechtstutzen der Bäume: Ein Baum enthält oft nach dem ersten Aufbau unnötige Unterbäume, die durch Überanpassung an die Trainingsdaten zu Stande kommen. Um diese Überanpassung zu beseitigen und den Entscheidungsbaum allgemeiner zu gestalten wird ein Teil der unnötigen Unterbäume in Blätter oder simplere Unterbäume zusammengefasst.

Das Zurechtstutzen erfolgt über zwei Methoden die beide rekursiv jeden Knoten bearbeiten. Bäume können entweder während dem Aufbau oder danach gestutzt werden. Stutzt man sie nach dem Aufbau, werden alle Knoten rekursiv mit einer der folgenden Methoden bearbeitet:

- **Unterbaumersetzung:** Der Unterbaum wird durch ein einzelnes klassifizierendes Blatt ersetzt, die Genauigkeit auf dem Trainingsset wird dadurch verringert. Da das Ziel die Verringerung der Überanpassung ist, sollte drauf geachtet werden, dass bei einem unabhängigen Testdatensatz die Genauigkeit erhöht wird.
- **Unterbaumerhebung:** Der Unterbaum wird eine Stufe nach zur Wurzel verschoben und ersetzt somit einen anderen Unterbaum. Die dadurch gelöschten Blätter werden in den erhobenen Unterbaum eingebaut.

Welche Methode angewendet wird, hängt von der *abgeschätzten Fehlerrate* auf dem unabhängigen Testdatensatz ab. Falls es keinen separaten Testdatensatz gibt, so ist es möglich einen Teil der Trainingsdaten zurückzuhalten und auf dem zurückgehaltenen Daten die Tests durchzuführen. Dies nennt man das *Kreuzvalidierungsverfahren*. Es lässt sich durch die mehrfache zufällige Auswahl der Trainingsdatensätze und den Durchschnitt aller Einzelfehlerraten zum *k-fachen Kreuzvalidierungsverfahren* erweitern.

Eine weitere Möglichkeit zur Auswahl der zu benutzenden Methode beim Zurechtstutzen der Bäume ist das *Zurechtstutzen nach Komplexitätskosten*. Hierbei wird statt nur auf die potentielle Fehlerrate zu achten, der Unterbaum zurechtgestutzt, der im Vergleich zu seiner Größe die kleinste Fehlerreduktion bringt.

Anwendungsgebiet: Entscheidungsbäume werden oft verwendet, um eine Vorauswahl zu treffen, welcher Algorithmus zum Bearbeiten des Problems am geeignetsten ist[Koh96]. Ein weiterer Anwendungszweck für Entscheidungsbäume liegt in der Medizin in der Erkennung von epileptischen Anfällen anhand von Gehirnstromaufnahmen[PG07].

2.2.2.5. Klassifikationsregeln

Regeln sind grundsätzlich den Entscheidungsbäumen ähnlich und beide lassen sich leicht in einander transformieren. Um eine Instanz mit Hilfe eines Regelsatzes zu klassifizieren, werden die einzelnen Regeln durchlaufen und getestet. Besteht eine Instanz alle Tests einer Regel, so wird diese einer bestimmten, von dieser Regel implizierten, Klasse zugeordnet.

Daraus wird ersichtlich, dass die Reihenfolge in der die Regeln abgefragt werden, durchaus Relevanz hat. Regeln die später abgefragt werden, sind bereits von vorherigen Regeln abgedeckt worden. Deshalb ist die Reihenfolge beim Trainieren der Regeln ebenfalls wesentlich.

Um die Regelsätze aus den Trainingsdaten zu inferieren wird ein *Abdeckungsansatz* angewendet. Es wird mit der Menge der noch zu klassifizierenden Instanzen begonnen. In jedem Durchlauf wird ein Teil der Instanzen rekursiv durch Regeln abgedeckt. Sollte die Menge der abgedeckten Instanzen nicht rein sein, so wird durch rekursive Erweiterungen der Regel diese Menge verkleinert, bis sie rein ist. Wird eine reine Menge erreicht so werden die Instanzen, die von dieser Regel klassifiziert werden, aus der Gesamtmenge aller noch zu klassifizierenden Instanzen entfernt. Es ist möglich, statt absolute Reinheit der Mengen zu verlangen, den Rekursionszweig abzubrechen, falls die Fehlerrate einen vorher definierten Wert unterschreitet, um Überanpassung zu vermeiden. Dies wird wiederholt, bis die Klassen der Gesamtmenge einheitlich sind. Diese Klasse wird als Standardklassifikationsregel definiert, die eintritt falls keine andere Regel auf die Instanz zutrifft.

Da beim Erstellen der Klassifikationsregeln, abhängig vom Kriterium nach welchem die Regeln hinzugefügt werden, unterschiedliche Regelsätze zustande kommen, ist es wichtig dies möglichst passend zu wählen.

Es gibt zwei Standardansätze zum Lernen neuer Regeln. Beide haben zum Ziel die Genauigkeit auf unbekannten Testdatensätzen zu erhöhen. Dabei wird der *Informationsgewinn*, das heißt so viele Instanzen wie möglich zu klassifizieren, maximiert. Die Abdeckung der Instanzen wird währenddessen minimiert. Der erste Ansatz optimiert am Ende durch einen globalen Schritt den gesamten Regelsatz. Der zweite Ansatz arbeitet während dem Trainieren und basiert auf dem wiederholten teilweisen Aufbauen und Stutzen von Entscheidungsbäumen. Der Entscheidungsbaum wird aufgebaut und der Weg zu dem klassifizierenden Blatt, welches die größte Abdeckung der Instanzen hat, wird zu einer Regel transformiert. Diese Instanzen werden aus der Ursprungsmenge entfernt und es wird ein weiterer Entscheidungsbaum aufgebaut. Da der wiederholte Aufbau ganzer Bäume ineffizient ist, werden für gewöhnlich nur Teilbäume aufgebaut. Diese Teilbäume enthalten Äste die undefiniert sind und deshalb ignoriert werden. Sobald ein stabiler Baum, der nicht weiter vereinfacht werden kann, gefunden wurde, kann von diesem die Regel abgelesen werden.

Anwendungsgebiet: Eine Unterform der Klassifikationsregeln, die *Assoziationsanalyse*, welche die Suche nach einer Korrelation zwischen den Attribut darstellt, findet oft Anwendung bei automatischen Kundenempfehlungen[CL01]. Dabei wird zwischen verschiedenen gekauften Produkten eine Assoziation hergestellt, die den Kunden, auf für ihn interessante Produkte, aufmerksam machen soll. Als einfaches Beispiel wäre die Empfehlung zum Kauf einer Speicherkarte, beim Kauf einer Digitalkamera.

2.2.2.6. Lineare Modelle

Lineare Modelle sind besonders geeignet für Instanzen die hauptsächlich von numerischen Attributklassen abhängig sind. Die Attribute lassen sich leicht, durch ihre numerische Eigenschaften, auf die Achsen eines kartesischen Koordinatensystems übertragen.

Die **Logische Regression** dient in der Statistik hauptsächlich dazu, von bekannten Attributen einer unbekannten Instanz auf unbekannte Attribute derselben Instanz schließen zu könnten. Sie kann aber auch zur Klassifikation verwendet werden. Dazu wird für jeden möglichen Klassenwert in der Trainingsmenge eine Regressionsfunktion, zum Beispiel mit der Methode der kleinsten Quadrate, berechnet. Eine unbekannte Instanz wird klassifiziert indem ihr die Klasse des am nächsten liegenden Wertes der Regressionsfunktion zugewiesen wird.

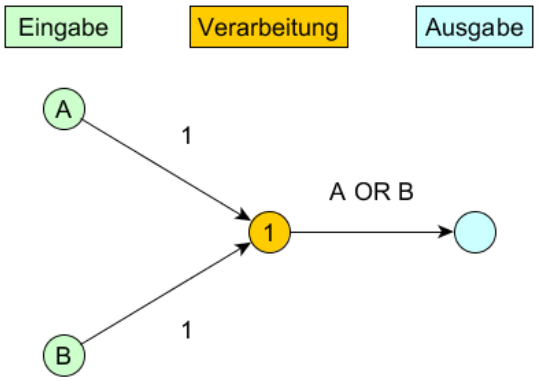

Abbildung 2.5.: Perceptron das OR entscheidet

Die Grundidee der **Klassifikation** anhand von linearen Modellen ist es den Raum durch die Definition einer Hyperebene den Raum aller Instanzen in eine binäre Klasse zu teilen. Jede neue unbekannte Instanz wird abhängig von ihrer Position im Raum, direkt durch die lineare Funktion definierte Hyperebene, klassifiziert. Zum Lernen dieser Gewichte kann ein Perceptron verwendet werden.

Das **einlagige Perceptron** ist ein einelementiges Neurales Netzwerk welches eine teilende Hyperebene findet, sofern diese existiert. Der Algorithmus trainiert das Perceptron iterativ, wobei nach jeder Iteration alle Instanzen auf korrekte Klassifikation geprüft werden, und falls die Instanz falsch klassifiziert wurde, die Gewichte angepasst werden. Das Perceptron klassifiziert, indem überprüft wird, ob ein bestimmter Wert überschritten wurde. Es besteht aus einem Ausgangsknoten, einem Knoten für die Berechnung und mehreren Eingangsknoten, die jeweils ein Attribut der Instanzen und einer Konstanten repräsentieren. Die Kanten der Eingangsknoten besitzen Gewichte die zu Beginn mit 0 Initialisiert werden. Das Perceptron lernt *fehlergetrieben* und ist ein linearer Klassifikator.

In Abbildung 2.5 ist ein einfaches einlagiges Perceptron graphisch dargestellt. Es feuert, das heißt setzt die Ausgabeklasse auf `WAHR`, falls mindestens einer der beiden Eingangsknoten den Wert 1 hat. Es berechnet somit die Logische `OR`-Funktion.

Eine Abwandlung des Perceptrons ist das **gewichtete Perceptron**. Anstatt wie beim einfachen Perceptron nur die zuletzt als falsch klassifizierte Instanz zur Gewichtsbestimmung zu nutzen, verwendet es auch die Instanzen davor. Dabei wird die Anzahl der vorher korrekt klassifizierten Instanzen dazu genutzt, die Wirkung der falsch klassifizierten Instanz auf das Gewicht zu verringern.

Da das Percepton beim Trainieren abhängig von verschiedenen Faktoren, wie die Reihenfolge der Instanzabfragen, ist, können unterschiedliche Modelle entstehen. Dies führt dazu, dass der Abstand vom linearen Klassifikator zu der nächstliegenden Trainingsinstanzen unterschiedlich groß ist. Benutzt man einen Klassifikator bei dem dieser Abstand zu gering ist, so kann es sein, dass neue Instanzen falsch klassifiziert werden. Die **Stützvektormaschinen (SVM)** lösen diese Problem, indem garantiert wird, dass der Abstand vom linearen Klassifikator zu der am nächsten liegenden Instanz immer maximiert wird. In Abbildung 2.6 wird der Unterschied zwischen einer SVM (grüne Linie) und einem einlagigen Perceptron (schwarze Linie) deutlich. Die Ränder der Klassifikatoren sind durch dünnere Geraden dargestellt und zeigen den maximalen Abstand zu den nächstgelegenen Instanzen. Während der Abstand beim Perceptron gering ist und es dadurch sehr leicht Fehler bei der Einteilung geben kann, wurde bei der SVM der größtmöglichste Abstand zwischen der Hyperebene und den Datenpunkten der Trainingsmenge erzeugt. Dadurch

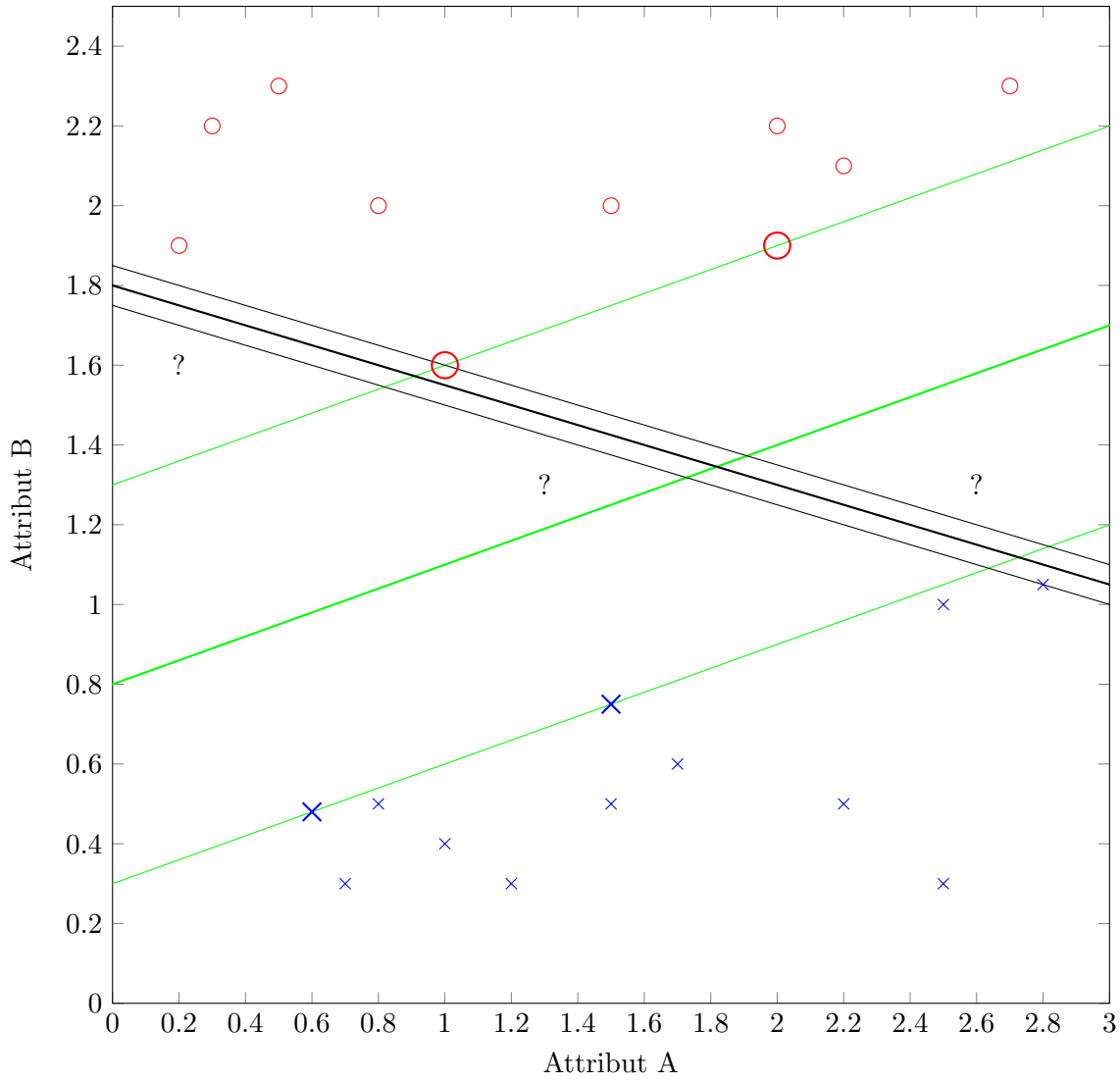

Abbildung 2.6.: Unterschied zwischen Perceptron (schwarze Linie) und SVM (grüne Linie). `?` stellt hierbei unbekannte, noch zu klassifizierende Instanzen dar.

können die die unbekannten Instanzen `?` sicherer klassifiziert werden als beim Perceptron. Dies führt auch dazu, das Perceptron und SVM unbekannte Instanzen `?` in unterschiedliche Klassen einteilen können. Wie auch aus der Abbildung ersichtbar, definieren nur die der Geraden am nächsten liegenden Instanzen den linearen Klassifikator. Nur diese *Stützvektoren* werden daher benötigt, um die Hyperebene mathematisch zu beschreiben. Die Stützvektoren der SVM sind in der Abbildung durch ihre Größe hervorgehoben. Die SVM ist ein *Breiter-Rand-Klassifikator* und findet die *Maximumabstandshyperebene*.

SVM und Perceptron, sind lineare Klassifikatoren und können daher nur linear trennbare Datenmengen korrekt klassifizieren. Zwar ist es möglich den Verfahren beim Trainieren zu erlauben eine bestimmte Anzahl an Instanzen falsch zu klassifizieren, um Ausreißer im Datensatz zu ignorieren, aber die gesamte Menge muss trotzdem weiterhin linear trennbar sein. Eine Erweiterung der SVM die dieses Problem beseitigt ist der **Kernel-Trick**. Hierbei werden die Daten in einen höher dimensionierten Raum transformiert, in welchem die Daten linear trennbar sind.

In Tabelle 2.3 wurde die `XOR`-Funktion in einen dreidimensionalen Raum erweitert, wobei die dritte Dimension eine Kombination der Attribute der Instanz ist. Anhand dieser dritten

Tabelle 2.3.: XOR-Funktion mit dritter Dimension

Eingabedimensionen			Klasse
a	b	(a+b)%2	c
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Dimension ist die Funktion linear klassifizierbar.

Ein großes Problem beim Erweitern und Transformieren des Attributraumes ist die stark steigende Berechnungskomplexität, die bei Datensätzen mit vielen Attributen die Klassifikation stark erschweren kann. Eine zu hohe Abhängigkeit von diesen künstlichen Attributen kann außerdem zu Überanpassung führen.

Mehrschichtiges Perceptron: Das Mehrschichtige Perceptron ist eine Erweiterung des einfach Perceptrons, in der nicht nur ein einziger Berechnungsknoten existiert, sondern mehrere Ebenen, die selbst aus mehreren Knoten bestehen. Die Ebenen sind miteinander verbunden. Beim Trainieren kann das Netzwerk entweder die Struktur der Knoten modifizieren, oder falls die Struktur bereits vorgegeben ist, die Gewichte der einzelnen Kanten.

Anwendungsgebiet: Das Mehrschichtige Perceptron zählt zu den Standardalgorithmen und wird oft zum Lösen komplexer Probleme eingesetzt. Als Beispiel wären die Bereiche der Sprach-[MB90] und Mustererkennung[MSB93] zu nennen. Da sie in im vergangenen Jahrhundert durch die SVM außer Mode geraten sind, erfahren sie seid 2009 durch das *Tiefenlernen*[HDY+12] und stärker werdender Computerhardware eine Renaissance. Dabei wird ein Netzwerk mit einer großen Anzahl an Ebenen trainiert.

2.2.2.7. Instanzbasiertes Lernen

Während die vorherigen Verfahren alle darauf beruhen anhand von Trainingsdaten ein Modell zu erstellen, welches unabhängig von den Trainingsdaten neue Instanzen klassifizieren kann, basieren die Instanzbasierten Methoden darauf, während der Klassifikation die Trainingsdaten zu prüfen und dabei die den unbekannten Instanzen ähnlichsten Trainingsinstanzen zu finden. Ein großer Vorteil dieses *trägen Lernansatzes* ist es, dass neue Instanzen nach der Klassifikation dem Trainingsdatensatz hinzugefügt werden können, um diesen somit zu verbessern.

Es ist offensichtlich, dass die Funktion, die die Distanz zwischen den Instanzen berechnet, einen hohen Anteil am Ergebnis spielt. Hierfür wird oft die euklidische Distanz verwendet. Eine alternative Methode die Distanz zu berechnen ist die Manhattenmetrik, bei der die Distanz zwischen zwei Instanzen, die Summe der absoluten Differenzen ihrer Einzelkoordinaten ist.

Um Instanzen mit unbekannte Werten klassifizieren zu können, werden diese als maximal unterschiedlich von allen anderen Instanzwerten definiert. Dadurch werden unbekannte Werte so groß wie möglich. Nicht numerische Werte werden hingegen, wie bei den meisten anderen Verfahren, bei Gleichheit als die Distanz von 0, und bei Ungleichheit als die Distanz von 1 definiert.

Damit Attributklassen mit kleineren Werteräumen nicht überbewertet werden, werden die Werteräume normalisiert. Eine Gewichtung der einzelnen Attribute kann ebenfalls stattfinden, falls bestimmte Attribute als irrelevant erachtet werden.

Es gibt verschiedene Ansätze und Methoden, bei denen die Suche effizienter gestaltet wird, ohne dabei das Ergebnis zu ändern. Ein Beispiel hierfür sind *kd-Bäume*. Sie sorgen dafür,

dass ein Teil der Instanzen, die garantiert nicht am nächsten zur zu klassifizierenden Instanz liegen, später geprüft werden. Dies ermöglicht eine schnellere Suche auf sehr großen Trainingsdatensätzen. Eine Alternative ist es eine Vorauswahl der Instanzen vorzunehmen. Dabei werden nur bestimmte Instanzen in den Datensatz, der beim Klassifizieren durchsucht wird, aufgenommen. Hierfür ist eine Methode, nur die Instanzen aufzunehmen, denen die falsche Klasse zugeordnet wurde. Bei besonders stark *rauschenden* Datensätzen führt dies aber dazu, dass wiederum sehr viele Instanzen gespeichert werden. Um das Rauschen generell zu reduzieren, werden meist mehrere Instanzen zur Klassifikation benutzt. Eine Glättung des Datensatzen kann durch das Verwerfen von Instanzen, anhand deren oft falsch klassifiziert wird, stattfinden.

Anwendungsgebiet: Der nächste-Nachbarn Algorithmus kommt durch seine Einfachheit in einer Vielzahl von verschiedenen Anwendungen vor. Als Beispiele sind die optische Buchstabenerkennung[MCS99] und das Vorschlagen von Wörtern bei einer Rechtschreibprüfung[Sav01] zu nennen.

2.3. Weka

Die „Waikato Environment for Knowledge Analysis“ (Weka) ist eine Softwareumgebung, die von der neuseeländischen Universität von Waikato entwickelt wurde[WFH11]. Weka stellt Methoden und Verfahren aus dem Bereich des maschinellen Lernens zur Verfügung. Datensätze können gefiltert werden, um so beispielsweise eine Dimensionsreduktion durchzuführen Das generelle Arbeiten mit Daten wird dadurch vereinfacht. Weka hilft unter anderem bei der Visualisierung von Daten, stellt Klassifikationsverfahren zur Verfügung und vereinfacht das Trainieren und Analysieren der Ergebnisse und Verfahren. Es ist in Java geschrieben und erlaubt es weitere Verfahren über Erweiterungsmodule hinzuzufügen.

In dieser Arbeit werden die bereits in Weka implementierten Klassifikationsverfahren genutzt. Über die Klassifikationsverfahren sollen Wortartmarkierer kombiniert werden. Außerdem werden die von Weka bereitgestellten Techniken zur Evaluation der Klassifikationsverfahren genutzt.

2.4. Evaluationsmethoden

Um eine Aussage über die Qualität verschiedener Klassifikationsverfahren zur Kombinierung von Wortartmarkierern treffen zu können, werden diese evaluiert. Dabei wird ein Korpus mit bekannter Lösung in Trainings- und Testdaten aufgeteilt. Auf dem Trainingskorpus wird durch das Klassifikationsverfahren ein Modell trainiert. Dieses Modell wird genutzt, um die Testdaten zu klassifizieren. Das Ergebnis der Klassifikation wird mit der korrekten Lösung verglichen und der Anteil an korrekt klassifizierten Daten festgestellt.

Das Ergebnis hängt stark von der Auswahl der Trainingsdaten ab. So kann es vorkommen, dass besonders leicht zu klassifizierende Daten in der Testmenge sind, was zu einer Überbewertung des Verfahrens führt. Andererseits kann es passieren, dass die ausgewählten Trainingsdaten, sich zu stark von den Testdaten unterscheiden. Dies führt dazu, dass das Verfahren unterbewertet wird. Da die Ausführung eines einzelnen Testes nicht aussagekräftig ist, werden mehrere Tests mit unterschiedlichen Testmengen durchgeführt.

Aus den Ergebnissen der Tests können drei Metriken abgeleitet werden. Die erste Metrik ist der Erwartungswert der Anteile an korrekt klassifizierten Daten:

$$E(X) = \sum_{i \in I} x_i p_i \tag{2.2}$$

Der Erwartungswert einer Zufallsvaribale X berechnet sich als Summe der Zufallszahlen x_i gewichtet mit ihren Auftrittswahrscheinlichkeiten p_i. Anhand des Erwartungswertes lässt

sich die Qualität eines Verfahrens feststellen. Die Abweichung vom Erwartungswert wird als Varianz bezeichnet:

$$Var\,(X) = E\left((X-\mu)^2\right), \tag{2.3}$$

Die Varianz berechnet sich als Erwartungswert E der quadrierten Different des Erwartungswert einer Zufallszahl $\mu = E\,(X)$ von der Zufallszahl X selbst. Dieser Wert stellt die Robustheit des Verfahrens dar. Die letzte Metrik ist Cohens Kappa:

$$\kappa = \frac{p_0 - p_c}{1 - p_c} \tag{2.4}$$

Es berechnet sich durch das Verhältnis zwischen der Differenz vom zufällig erwarteten Übereinstimmung p_c zum Übereinstimmungswert der beiden Schätzer p_0 und der Substraktion der zufällig erwarteten Übereinstimmung p_c von 1. Cohens Kappa ist das Ausmaß der Übereinstimmung der Einschätzungsergebnisse bei unterschiedlichen Beobachtern. Dadurch kann die Objektivität der Kreuzvalidierung angegeben werden.

Es gibt mehrere Testverfahren, von denen zwei besprochen werden sollen. Beide liefern ähnliche Werte beim Erwartungswert, der Varianz und Cohens Kappa. Unterschiede im Ergebnis entstehen durch die zufällige Auswahl der Instanzen. Beide Verfahren sind Kreuzvalidierungsverfahren und führen Tests mehrfach aus. In Molinaro et al.[MSP05] werden beide Verfahren beschrieben und evaluiert.

2.4.1. k-fachen Kreuzvalidierung

Bei der k-fachen Kreuzvalidierung wird der gesamte Datensatz zufällig in k gleich große Teilmengen aufsplittet. Nun wird reihum eine Teilmenge als Testmenge definiert. Auf den andern k-1 Teilmengen wird ein Modell trainiert. Das Modell wird auf der ausgewählten Testmenge evaluiert. Dies wird wiederholt bis auf allen k Testmengen evaluiert wurde. Aus dem Durchschnitt der Ergebnisse der Evaluationen kann eine Schätzung der Erwartungswerte, Varianz und anderen Metriken abgegeben werden.

Ein großer Vorteil dieser Methode ist das garantiert ist, dass auf jeder Instanz gestest und trainiert wurde. Auch wird auf jeder Instanz genau ein einziges mal getestet. Ein Nachteil ist aber, dass nicht alle möglichen Kombinationen durchgeführt werden. Ein weiterer Nachteil ist, dass die Größe der Trainings- und Testmenge abhängig von der Anzahl der Wiederholungen ist. Die kleinste mögliche Trainingsmenge ist daher 50% der Gesamtmenge. Dies ist bei der 2-fachen Kreuzvalidierung der Fall.

2.4.2. Monte-Carlo-Kreuzvalidierung

Bei der Monte-Carlo-Kreuzvalidierung wird der Datensatz zufällig in eine Trainings- und Testmenge aufgetrennt. Die Anteile der Trainings- und Testmengen können hierbei festgelegt werden. Aus der Trainingsmenge wird ein Modell trainiert, welches auf der Testmenge evaluiert wird. Nun wird das Prozedere mehrfach wiederholt und der Durchschnitt der Ergebnisse berechnet. Die Anzahl an Wiederholungen kann zu Beginn festgelegt werden und eine Erhöhung führt zu einem genaueren Ergebnis.

Der wichtigste Vorteil dieser Methode ist dass die Anteile für die Trainings- und Testmengen frei bestimmbar sind. Ein Nachteil ist, dass, besonders bei extremen Anteilsverhältnissen, bestimmte Instanzen nie als Test- oder Trainingsmenge genutzt werden. Durch Erhöhung der Wiederholungen kann hier aber entgegengewirkt werden. Dies führt aber dazu, dass auf manchen Instanzen mehrfach getestet wird. Auch hier werden nicht alle möglichen Kombinationen getestet, aber mit Erhöhung der Wiederholungsanzahl wird sich dem angenähert.

2.5. Zusammenfassung

Die in diesem Kapitel beschriebenen Grundlagen gaben einen Überblick über die Funktionsweise für die Kombination genutzten Klassifikationsalgorithmen. In späteren Kapiteln werden die hier beschriebenen Verfahren und Grundbegriffe wiederverwendet. Die erläuterten Grundbegriffe der Wortartmarkierung werden zur Beschreibung der verwendeten Wortartmarkierer genutzt. Die Klassifikationsalgorithmen werden bei der Kombination dieser Wortartmarkierer verwendet. Auch wurde ein kurzer Überblick über Evaluationsverfahren und die Weka-Softwareumgebung gegeben.

3. Verwandte Arbeiten

In dieser Arbeit soll untersucht werden, ob Wortartmarkierer kombiniert werden können, sodass eine Ergebnisverbesserung stattfindet Wortartmarkierer sind Klassifikationsprogramme, die in der Computerlinguistik eingesetzt werden um Wörtern eines Satzes die entsprechenden Wortarten zuzuweisen. Es gibt daher drei verwandte Themengebiete, aus welchen verwandte Arbeiten verwendet werden können. Dieses Kapitel wird deshalb in drei Abschnitte eingeteilt. Abschnitt 3.1 behandelt Arbeiten, die sich mit den Möglichkeiten der Kombinationen von Klassifikatoren auseinandersetzen. Eine Annäherung an das Thema dieser Arbeit findet dadurch statt, dass in Abschnitt 3.2 Arbeiten betrachtet werden, die Klassifikationsverfahren im Bereich Computerlinguistik kombinieren. Dabei lassen sich viele der allgemeinen Methoden und Ideen auf die Wortartmarkierung übertragen. Im letzten Abschnitt 3.3 werden verwandte Arbeiten vorgestellt, die Wortartmarkierer kombinieren. Neben Arbeiten, bei denen die Genauigkeit der Wortartmarkierer durch Kombination verbessert wird, werden auch Arbeiten erläutert, die die Kombinierung von Wortartmarkierer für alternative Zwecke nutzen oder zur Wortartmarkierung von Sprachen mit geringer Korpusgröße.

Abschließend wird aus den Arbeiten in Abschnitt 3.4 eine Schlussfolgerung gezogen.

3.1. Allgemeines zur Kombination

Die in diesem Abschnitt vorgestellten Arbeiten sind grundlegender Natur und dienen daher als generelles Beweismittel. Sie zeigen, dass es grundsätzlich möglich, ist mehrere Klassifikationsverfahren zu kombinieren, um ein besseres Ergebnis zu erreichen als die besten verwendeten Einzelklassifikatoren.

Bereits 1996 zeigten Ali und Pazzani[AP96], dass die Kombination verschiedener Klassifikatoren ein besseres Ergebnis als die Einzelklassifikatoren erreichen kann. Das Ausmaß der Verbesserung hängt dabei davon ab, wie stark die einzelnen Klassifikatoren korreliert sind. Schwach korrellierte Klassifikatoren führen hierbei zu einem besseren Ergebnis als welche die ähnliche Fehler machen. Klassifikatoren können sich also gegenseitig korrigieren.
Es wird auch gezeigt, dass die Qualität des Lernmaterials von Bedeutung ist. Sind die Trainingsdaten verrauscht, das heißt ungenau und enthalten Fehler, so führt das zu schlechterer Performanz als das Training auf rauschfreien Daten. Die Ursache hierfür ist der fehlende Informationsgehalt von verrauschten Daten. Daraus folgt, dass eine schlechte Wahl der Attribute oder Klassifikationsverfahren, dazu führen kann, dass die Performanz sinkt. Es ist daher wichtig möglichst nur kompetente Klassifikatoren auszuwählen.

Das es nicht nur möglich ist verschiedene Klassifikationsverfahren zu kombinieren, sondern auch unterschiedliche Trainingskorpora zeigten Freund und Schapire[FS99]. Sie beschreiben das Prinzip der *Verstärkung* anhand dem von ihnen entwickelten Algorithmus Adaboost. Dabei wird ein schwacher Klassifikator sequentiell so trainiert, dass bei jeder Iteration mehr Gewicht auf die Klassifikation der in der letzten Iteration falsch klassifizierten Instanzen gelegt wird. Durch Kombination der Sequenz aus schwachen Klassifikatoren zu einem starken Klassifikator kann eine Leistungssteigerung gegenüber den einzelnen schwachen Klassifikatoren erreicht werden. Dies zeigt, dass es möglich ist die gleichen Klassifikationsverfahren zu kombinieren, sofern sie unterschiedliche Modelle nutzen. Auf Wortartmarkierer übertragen, bedeutet dies, dass es möglich ist Markierer zu kombinieren, die das gleiche Verfahren nutzen, aber auf unterschiedlichen Korpora trainiert wurden.

3.2. Kombination in der Computerlinguistik

Da Wortartmarkierung ein Teilbereich der Computerlinguistik ist, kann es sinnvoll sein allgemeine Konzepte zu betrachten. Viele generelle Verfahren lassen sich auf die Wortartmarkierung übertragen. In diesem Abschnitt sollen daher zwei Arbeiten betrachtet werden, die sich mit der Verarbeitung von natürlicher Sprache beschäftigen. Beide Arbeiten kombinieren mehrere Klassifikatoren um ein neues Ergebnis zu klassifizieren, nutzen dafür aber nicht explizit Wortartmarkierer.

Im Jahr 2011 traten IBM (Research) mit Watson[FBCC$^+$10] bei der Quizshow Jeopardy an. Damit ein Computer Fragen sicher beantworten konnte, mussten mehrere Probleme gelöst werden. Als erstes musste Watson die vom Moderator gestellte Frage verstehen, dann mögliche Antworten finden und die entsprechenden Konfidenzen für die Antworthypothesen in Echtzeit berechnen, um zu entscheiden ob eine Antwort gegeben werden kann. Da nicht erwartet werden kann, dass eine einzelne Komponente für jedes Problem allein ein perfektes Ergebnis liefert, wird eine Kombination verschiedener Klassifikatoren benutzt, um Konfidenzen zur Antwortauswahl zu berechnen. Es wurden mehr als hundert verschiedene Techniken in der DeepQA-Architektur parallel zum Lösen der entsprechenden Bereiche genutzt. Watson setze sich erfolgreich gegen zwei menschliche Mitspieler durch. Dies zeigt, dass es möglich durch die Kombination nicht nur ein einziges Ergebnis zu bekommen, sondern mehrere Ergebnisse mit verschiedenen Konfidenzen.

Im Teilbereich der Klassifizierung der semantischen Bedeutung mehrdeutiger Wörter (engl. Word Sense Disambiguation) von natürlicher Sprache kombiniert Pederson[Ped00] mehrerer Bayes-Klassifikatoren, um die Genauigkeit zu verbessern. Hierbei werden neben dem Wort dem einzelnen Bayes-Klassifikatoren zwei Wortbitvektoren übergeben. Ein Bitvektor für die linke Seite, zeigt an, ob ein bestimmte Wörter Vorgänger des untersuchten Wortes sind. Der zweite Bitvektor zeigt dies, entsprechend für die Nachfolger des Wortes an. Die Länge der Wortbitvektoren ist abhängig davon wie viele Wörter sie abdecken und wird in drei Kategorien (Kurz, Mittel, Lang) eingeteilt. Es werden nun Bayes-Klassifaktoren, abhängig von ihren Kategorien, mit beidseitig verschieden langen Wortbitvektoren trainiert und auf ihre Performanz untersucht. Die besten Bayes-Klassifikatoren aus ihrer jeweiligen Kategorie werden kombiniert. Evaluiert wird dieser Zusammenschluss auf Erkennung der korrekten semantischen Bedeutung von zwei bestimmten Wörtern in einem hierfür entworfenen Korpus. Auf dem englischen Wort „*line*“ erreicht der beste Einzelklassifikator eine Genauigkeit von 84% während der Zusammenschluss dieses Ergebnis auf 88% verbessert. Beim englischem Wort „*interest*“ erhöht der Zusammenschluss das Ergebnis von 86% auf 89%. Da die semantische Bedeutung Einfluss auf die Wortart des Wortes hat, könnte eine ähnliche Vorgehensweise Wortartmarkierer verbessern. Aus der wahrscheinlichsten Bedeutung eines Wortes könnte bei Mehrdeutigkeit der Wortart die korrekte Markierung abgeleitet werden.

3.3. Kombination von Wortartmarkierern

Neben der allgemeinen Kombination von Klassfikationsalgorithmen, wurde auch an der Verbesserungen von Wortartmarkierern durch Kombination geforscht. Es gibt hierbei viele Einsatzmöglichkeiten. Der größte Teil dieses Abschnittes beschäftigt sich mit der Ergebnissverbesserung durch Kombinierung der Wortartmarkierer. Die Methoden lassen sich daher meist direkt auf diese Arbeit übertragen und mit ihr vergleichen. Da die meisten Arbeiten sich mit der englischen Sprache beschäftigen, sollen am Ende einige Methoden vorgestellt werden, die für andere Sprachen entwickelt wurden.

3.3.1. Verbesserung der Genauigkeit

Da Wortartmarkierer ein Spezialfall der Klassifikation sind, lassen sich viele Verfahren, mit denen Klassifikatoren kombiniert werden können, auf diese übertragen. Als Erste nutzen daher Brill und Wu[BW98] im Bereich der Kombination von Wortartmarkierern verschiedene Methoden, um vier Markierer zu verbinden. Die genutzten Markierer waren ein Unigrammmarkierer, ein N-Gramm-Markierer, ein transformationsbasierter Markierer und ein Maximumentropiemodell, welcher mit einer Genauigkeit von 96,8% den besten Einzelmarkierer darstellte. Bereits die Methode der einfachen Mehrheitswahl setze sich gegenüber dem Maximumentropiemodell durch und reduzierte den Fehler um 6,9% auf eine Genauigkeit von 97,0%. Weitere erfolgreiche Methoden waren die Inferenz von Regeln zum Markieren von Wörtern abhängig vom Ergebnis der Einzelmarkierer (97,1%) und die Inferenz von Regeln welchem Wortartmarkierer vertraut werden soll (97,2%). Damit wurde gezeigt, dass es möglich ist nicht nur allgemeine Klassifikationsverfahren zu kombinieren, sondern gezielt eine Kombination von Wortartmarkierer zur Ergebnisverbesserung zu nutzen.

Ebenfalls beschäftigte sich van Halteren[vZD01] mit der Verbesserung verschiedener Wortartmarkierer durch Kombination. Es werden vier verschiedene Wortartmarkierer für drei verschiedene Korpora trainiert. Von den untersuchten Methoden schneidet das gewichtete Wahrscheinlichkeitswahlverfahren mit Wortkontext am Besten ab und erreicht auf dem WSJ-Korpus eine Fehlerreduktion von 11,3% und eine Erhöhung der Genauigkeit von 96,88% des Maximumentropiemarkierers auf 97,23%. Auf dem LOB-Korpus wird hingegen eine höhere Fehlerreduktion von 24,3% erreicht. Die Verbesserung der Genauigkeit vom VMM-Markierer von 97,55% auf 98,14% durch Kombination ist auch generell höher als beim WSJ-Korpus. Halterer gibt als Grund hierfür die höhere Inkonsistenz des WSJ-Trainingskorpus und Goldstandards an.

In einer weiteren Arbeit kombiniert van Halteren[vZD98] vier Wortartmarkierer mit Hilfe von verschiedenen Wahlverfahren. Es werden ein VMM-Markierer, ein transformationsbasierter Markierer, ein instanzbasierter Markierer und ein Maximumentromiemarkierer verwendet. Alle untersuchten Wahlverfahren zum Kombinieren der Markierer führten zu einer Verbesserung. Das paarweise Wahlverfahren, bei dem alle Markierer in Paaren gemeinsam für Markierungen stimmen, schneidet am besten ab. Hierbei werden abhängig von den Markierungen zweier Wortartmarkierer die Konfidenzen für verschiedene Markierungen berechnet. Die Stimme des Paares entspricht den berechneten Konfidenzen. Es kann dabei eine Markierung ausgewählt werden, die kein Markierer hat, aber trotzdem die höchste Konfidenz besitzt. Es erreicht eine Fehlerreduktion von 19,1% gegenüber dem Maximumentropiemarkierer mit 97,43% und kommt somit auf 97,92%.

Die Arbeit von Glass und Bangay[GB05] ist dieser Arbeit sehr ähnlich und weist wohl die meisten Übereinstimmungen mit dieser Arbeit auf. Wie auch bei allen anderen Arbeiten basiert die Idee darauf, dass die Fehler der Wortartklassifikatoren korrigiert werden können, sofern sie nicht korreliert sind.

Ein starker Überschneidungspunkt ist der Wechsel der Bereichs. Während die Wortartmarkierer auf dem WSJ-Korpus trainiert wurden, fand die Evaluation und das Training der Klassifikatoren auf dem SUSANNE-Korpus statt. Dies entspricht der in dieser Bachelorarbeit durchgeführten Arbeitsweise, wobei statt dem SUSANNE-Korpus neben dem WSJ-Korpus zwei weitere eigene Korpora mit noch geringerer Größe verwendet werden.

Während in dieser Arbeit das Problem der notwendigen manuellen Übersetzung zwischen den beiden Wortartmarkierungsmengen durch Verwendung der gleichen Wortartmarkierungsmenge umgangen wird, so mussten Glass und Bangay zwischen den beiden Korpora von Hand übersetzen. Es wurde deshalb analysiert ob und wie stark ein Wechsel der Korpora zwischen Test- und Trainingskorpus die Genauigkeit verringert. Durch das Übersetzungprozedere fiel die Genauigkeit der verschiedenen Markierer auf ungefähr 70% bei Verwendung aller Markierungen. Erst als die Wortartmarkiergunsmenge weiter verallgemeinert und auf nur 19 Hauptmarkierungen verkleinert wurden, konnte eine Genauigkeit von ungefähr 93,3% erreicht werden, was immer noch unter den auf dem WSJ-Korpus angegebenen Genauigkeiten liegt. Ein weiterer Prozentpunkt konnte durch das Abbilden aller „*to*“ auf ein spezielles TO-Markierungssymbol abgerungen werden.

Es wurden fünf diverse Wortartmarkierungsalgorithmen aus verschiedenen Bereichen untersucht und mit dem Ziel einer Verbesserung der Genauigkeit durch vier Methoden kombiniert. Interessanterweise konnte sich keine Kombinationsmöglichkeit gegenüber dem TreeTagger, welcher als bester Einzelmarkierer 94,53% erreichte, durchsetzen. Die erfolgreichste Methode, die optimal gewichte Mehrheitswahl, erreichte eine Genauigkeit von 93,95% unter Verwendung aller fünf Markierer. Durch das Entfernen von schlechteren Markierern, näherte sich die Genauigkeit dem besten Einzelmarkierer auf bis zu 94,43% an.

Einen anderen Ansatz zur Kombination wählten Khin und Aung[KA15], um zu zeigen, dass reine N-Grammwortartmarkierer dadurch verbessert werden können, indem Unterstützungsmarkierer genutzt werden. Dabei arbeiten Wortartmarkierer nicht parallel sondern sind hintereinander geschaltet. Trifft ein höher dimensionierter N-Grammmarkierer auf eine Sequenz, die er nicht zuvor im Trainingskorpus gelernt hat, so kann er die Aufgabe an einen (N-1)-Grammwortartmarkierer weiterreichen. Dieser niedriger dimensionierte Markierer hat, da er unspezifischere Sequenzen erlernt hat, eine höhere Chance die gesuchte Sequenz zu kennen und somit das Wort zu markieren. Bei Ausfall dieses Markierers kann das Prozedere rekursiv fortgeführt werden. Ist der Unigrammmarkierer erreicht so wird die Wortart nur noch nach der Wortartfrequenz des Wortes im Trainingskorpus ausgewählt.

Die Hauptaufgabe von Wortartmarkierern ist es die Wortarten von unmarkierten Texten zu bestimmen. Das eine Kombinierung von Wortartmarkierern nicht nur dazu genutzt werden kann, sondern auch zur Fehlerfindung in Trainingskorpora eingesetzt werden kann, zeigt die Arbeit von Berthelsen und Megyesi[BM00]. Sie beschäftigen sich damit ob durch die Kombination von Markierern falsch markierte Wörter im Trainingskorpus selber gefunden und korrigiert werden können. Es wird also die Rauschunterdrückung durch Kombination mehrerer Klassifikatoren versucht. Sie erzeugen zuerst einen verrauschten Korpus. Bei Wortarten, bei denen eine hohe Mehrdeutigkeit vorhanden ist, wird durch Zufall eine der Alternativen gewählt. Dies wird zum Erzeugen verschiedener Verrauschungsgrade wiederholt. Auf den verrauschten Korpora werden nun drei Wortartmarkierer mit fünf Modellen trainiert. Diese werden mit zwei Methoden kombiniert um das Rauschen auf dem gleichen Korpus zu reduzieren. Beide Kombinationsmethoden führen zu einer Rauschreduktion, unterscheiden sich aber darin wie häufig sie die Markierung ändern. So führt die Methode der Mehrheitswahl bei einem niedrigen Verrauschungsgrad dazu, dass bis zu 88,4% des eingebauten Rauschen korrigiert wird, aber zeitgleich 9,6% korrekte Markierungen falsch markiert werden. Die Konsensus-Methode, bei der eine Markierung nur dann gewählt wird, wenn alle Markierer übereinstimmen, führt zu einer Rauschreduktion von 75,9% und einer

Falschkorrektur von ungefähr 3,4%. Steigt der Verrauschungsgrad so führt dies bei beiden Methoden dazu, dass die Anzahl der korrekt korrigierten Wörter sinkt und die Falschkorrekturen steigen. Anstatt also die Wortartmarkierer direkt zu kombinieren, werden mehrere Wortartmarkierer genutzt um den Korpus zu korrigieren. Dadurch kann auf dem Korpus ein neuer Markierer trainiert werden der besser ist als die ursprünglichen Modelle.

3.3.2. Weitere Sprachen

Da für viele andere Sprachen nur ein geringer Korpus existiert, wird oft versucht durch alternative Methoden die Genauigkeit zu steigern. Während in der englischen Sprache mehrere Korpora aus verschiedenen Bereichen existieren, ist dieser Luxus bei anderen Sprachen nicht gegeben. Da das Erstellen eines großen Korpus viel Arbeit beansprucht, wird oft versucht durch Kombination von Wortartmarkierern die Genauigkeit zu steigern. Da in dieser Arbeit zwei Korpora von geringer Größe verwendet werden, lassen sich diese Methoden übertragen.

Für brasilianisches Portugiesisch kombinierten Aires et al.[Rac] vier Wortartmarkierer mit zwölf Methoden. Der beste Einzelmarkierer war das Maximumentropiemodell mit 88,73% Genauigkeit. Während viele ihrer Kombinationsmethoden ein schlechteres Ergebnis lieferten als der beste Einzelmarkierer konnten vier Methoden eine Verbesserung erreichen. Die beste dieser Methoden, eine spezielle Form der Mehrheitswahl, erreichte eine Genauigkeit von 89,42%. Obwohl ein positives Ergebnis vorliegt, ist die Genauigkeit für praktische Anwendungen immer noch zu gering und die Kombination von Wortartmarkierern einen größeren Trainingskorpus nicht ersetzen kann.

Für Schwedisch evaluiert Sjöbergh[Sjö03] die Kombination von sieben verschiedene Wortartmarkierungsalgorithmen, wovon der beste Einzelmarkierer eine Genauigkeit von 95,9% erreicht, durch verschiedene Kombinationsmethoden. Die beste dieser Methoden, ein instanzbasiertes Lernverfahren welches auch die Konfidenzen nutzt, erreicht eine Genauigkeit von 96,7%. Die einfache Mehrheitswahl erreicht diese Genauigkeit knapp mit 96,6%. Die Fehler, die weiterhin übrig sind, werden von den meisten Markierern wiederholt. Dies bedeutet, dass vor allem unübliche Einzelfehler korrigiert wurden. Als weiterer Optimierungsschritt wird daher vorgeschlagen, dass eine weitere Fehlerreduktion, durch die Generierung von Regeln stattfinden kann. Durch vier handgeschriebene Regeln konnten 131 Fehler korrigiert werden, während nur 32 neue eingeführt wurden.

Für Italienisch nutzt Søgaard[Søg] Entscheidungsbäume und Nächste-Nachbar-Methoden und kombiniert so drei Wortartmarkierer. Als Einzelmarkierer wurden ein Maxiumumentropiemodell, der transformationsbasierte Markierer von Brill und der TreeTagger verwendet. Während der beste Einzelmarkierer eine Genauigkeit von 90,75% erreicht, verbessert die Nächste-Nachbarn-Methode die Genauigkeit auf 94,78%. Entscheidungsbäume erreichen eine Verbesserung auf 94,41%. Bei unbekannten Wörtern, bei denen der beste Einzelmarkierer nur eine Genauigkeit von 76,00% erreicht, erreicht die Methode der Entscheidungsbäume eine Steigerung auf 90,36%.

Da für die polnische Sprache nur ein kleiner markierter Trainigskorpus existiert, nutzt die TaKIPI-Architektur von Piasecki und Wardyski[Mac] eine Vielzahl an regelbasierten Klassifikatoren, welche in drei Schichten sequenziell ausgeführt werden. Jede Schicht übernimmt einen kleinen Teil der Klassifikation. Kombinationsmethoden, wie die Berechnung der durchschnittlichen Wahrscheinlichkeit der Klassifikatoren, stellen den hinterher folgenden Schichten ein Ergebnis zur Verfügung. Die folgenden Schichten verfeinern dieses Ergebnis weiter. Damit kann eine Steigerung vom besten Einzelmarkiererergebnis von 93,11% auf durchschnittlich 93,53% durch Kombination erreicht werden.

3.4. Fazit

Aus den verschiedenen Arbeiten folgt, dass eine Optimierung der Wortartmarkierer durch Kombination möglich ist. Die meist verwendete Kombinationsweise ist das einfache Wahlverfahren mit nur wenigen Wortartmarkierern. Bereits einfache Klassifikationsverfahren zeigten sich aber oft diesem Wahlverfahren überlegen. Es soll daher überprüft werden ob andere Klassifikationsalgorithmen zur Kombination genutzt werden können und ob eine größere Verbesserung als das einfache Wahlverfahren möglich ist und wie groß diese Verbesserung ist. Weiterhin nutzen die Arbeiten meist nur eine geringe Anzahl an Wortartmarkierern, obwohl mehr Wortartmarkierer mehr Informationen liefern. Auch wenn eine wahllose Erhöhung der Wortartmarkierer zu einem negativen Ergebnis führen kann, soll die Anzahl der genutzten Wortartmarkierer deshalb erhöht werden. Durch Verwendung von komplexen Klassifikationsverfahren, soll dieser Nachteil minimiert werden. Dadurch sollen die Vorteile von weiteren Markierern überwiegen.

4. Analyse und Entwurf

Für viele Anwendungen in der Computerlinguistik ist es nötig, aus Sätzen die Wortarten der Worte zu extrahieren. Wortartmarkierer sind Programme, die diese Aufgabe erfüllen. Ein Beispiel für eine solche Anwendung ist das Generieren von Bildern und Animationen aus natürlicher Sprache. Dabei ist die Klassifizierung von Wortarten einer der ersten Schritte in der Computerlinguistik. Falsch klassifizierte Wortarten führen dabei zu Folgefehlern, die auf den gesamten Ablauf Auswirkungen haben. Es ist also wichtig, die Anzahl der Fehler bei der Wortartmarkierung möglichst gering zu halten.

Da die meisten Wortartmarkierer auf dem WSJ-Korpus (siehe Abschnitt 4.2) trainiert werden, sinkt die Genauigkeit auf anderen Korpora. Eine Möglichkeit, eine Verbesserung zu erreichen, ist das Training auf einem dem Anwendungsfeld sprachlich möglichst ähnlichen Korpus. Es existiert aber nicht für jedes Anwendungsfeld ein passender Korpus ausreichender Größe. Das Training auf kleinen Korpora führt dabei nur zu geringen Verbesserungen oder sogar zu Verschlechterung. Eine weitere Möglichkeit, die Anzahl an Fehlern zu reduzieren, ist es, mehrere Wortartmarkierer zu kombinieren. Da sich Wortartmarkierer in ihrer Funktionsweise unterscheiden führt dies zu unterschiedlichen Fehlern. Dadurch ist es grundsätzlich möglich, dass Wortartmarkierer sich gegenseitig unterstützen und korrigieren. In Kapitel 3 finden sich einige verwandte Arbeiten, bei denen dieser Ansatz verfolgt wurde. Zur Kombinierung wurde dabei meist die einfache Mehrheitswahl verwendet. Dabei wird das Wort mit der Markierung versehen, die am öftesten von den Wortartmarkierern vorgeschlagen wurde. Obwohl dies das simpelste Kombinationsverfahren ist, konnte bereits in vielen Arbeiten eine Verbesserung erreicht werden. Die einfache Mehrheitswahl weist aber einige Probleme auf. So wird allen Wortartmarkierern das gleiche Gewicht zugewiesen, was dazu führt, dass der Einfluss der guten und schlechten Wortartmarkierer auf das Ergebnis identisch ist. Diese Probleme können durch die Nutzung von komplexeren Verfahren minimiert oder sogar beseitigt werden. In dieser Arbeit soll daher untersucht werden, welche Verbesserung der Ergebnisse erreicht werden kann, indem mehrere Wortartmarkierer durch Klassifikationsverfahren kombiniert werden.

Ein großer Teil dieses Kapitels ist die Analyse und Aufarbeitung der Problemstellung der Wortartmarkiereroptimierung. Hierbei soll die Problemstellung von einem großen abstrakten Problem in kleine konkrete Teilprobleme zerlegt werden. Die Lösungen der Teilprobleme führen wiederum dazu, dass ein Entwurf entsteht, der die abstrakte Problemstellung löst.

Für den Vergleich verfügbarer (z.T. bereits trainierter) Wortartmarkierer und die Evaluation der Kombinierung der Wortartmarkierer wird ein eigener Goldstandard verwendet.

Dieser Goldstandard wird in Abschnitt 4.2 vorgestellt. Die Grundlage der Kombination bilden die Wortartmarkierer. In Abschnitt 4.3 werden Wortartmarkierer, die zur Weiterverwendung aus einer Gruppe verschiedener recherchierter Wortartmarkierer ausgewählt werden, besprochen.

Der Rest des Kapitels beschäftigt sich mit einem softwaretechnischen Entwurf zur Lösung der Problemstellung. Hierfür werden in Abschnitt 4.4 Ziele für den Entwurf definiert. Die zur Erfüllung nötigen Strukturen (Unterabschnitt 4.4.1) und Formate (Unterabschnitt 4.4.2) werden in den darauffolgenden Abschnitten behandelt.

Aus dem Entwurf folgt, dass zwei Teilprogramme die definierten Probleme lösen sollen. Die Entwürfe für die Teilprogramme finden sich in Unterabschnitt 4.4.3 und Unterabschnitt 4.4.4.

4.1. Analyse der Problemstellung

Zum Lösen der Problemstellung wird die Möglichkeit untersucht, die Ergebnisse von Wortartmarkierern durch Kombination zu verbessern. Im Verbesserungsfall wird auch die Größe dieser festgestellt. Zur Kombination sollen Klassifikationsverfahren verwendet und die Qualität der Verfahren für die Kombination der Wortartmarkierer bestimmt werden. Die Ergebnisse der einzelnen Wortartmarkierer sollen hierbei als Attribute für die Klassifikationsverfahren dienen.

Damit lässt sich die Problemstellung in zwei Teilprobleme spalten, die unabhängig voneinander bearbeitet werden können. Das erste Teilproblem ist es, verschiedene Wortartmarkierer so zu verbinden, dass die Ergebnisse aller Wortartmarkierer weiterverwendet werden können. Das zweite Teilproblem befasst sich mit der Verwendung dieser Ergebnisse als Eingabe für Klassifikationsverfahren, um so die einzelnen Wortartmarkierer zu kombinieren. Beide Teilprobleme sind, bis auf die Nutzung der Ergebnisse des ersten Teilproblems als Eingabedaten für das zweite Teilproblem, voneinander unabhängig. Sie werden daher separat betrachtet und gelöst.

Neben der zentralen Problemstellung gibt es einige Nebenprobleme. Zum einen müssen entsprechende Korpora ausgewählt werden, auf denen trainiert wird. Aufgrund der weiten Verbreitung der englischen Sprache sind die meisten Wortartmarkierer auf diese spezialisiert. Sie liefern meist ein Modell bei, dass auf dem *WSJ*-Korpus trainiert wird. Es werden keine eigenen Modelle trainiert, sondern die vom Entwickler trainierten Modelle verwendet. Dadurch wird überprüft, ob die Kombination nicht angepasster Modelle besser abschneidet als das beste Modell.

Für den Goldstandard werden zwei kleinere Korpora und ein Teil des WSJ-Korpus verwendet. Diese werden dazu genutzt, um die Klassifikationsverfahren zu trainieren und die Qualität der Markierungen zu bewerten.

4.2. Goldstandard

Um die Qualität der verschiedenen Klassifikationsalgorithmen und trainierten Modelle einschätzen zu können ist es nötig, das Ergebnis der Klassifikation mit einem korrekten Goldstandard und anderen Klassifikatoren zu vergleichen. Anhand eines Goldstandards lassen sich auch neue Modelle trainieren.

Im Fachbereich der Wortartmarkierungen besteht dieser Goldstandard aus Sätzen, bei denen die Worte mit ihren korrekten Wortartmarkierungen gespeichert sind. Das Format ist nicht standardisiert und unterscheidet sich daher nach Hersteller des Wortartmarkierers. Für diese Arbeit wird das Wort von der Wortartmarkierung durch ein „/“ getrennt.

4.2.1. Erzeugung des Goldstandards

Der Goldstandard wird in einem halbautomatischen Verfahren erstellt. Die Korpora werden durch den Wortartmarkierer von Stanford und die drei trainierten Modelle vormarkiert. Die Tokenisierung findet hierbei automatisch durch den Stanford-Tokenisierer statt. Die Unterschiede zwischen den Ergebnissen der Modelle werden zur Unterstützung sichtbar gemacht.

Nach den automatischen Schritten werden die Texte komplett von Hand korrigiert. Die Unterschiede zwischen den Modellen dienen hierbei als Unterstützung, hauptsächlich indem Alternativen zur falschen Wortartmarkierung angezeigt werden.

Viele Fehler werden aber von allen Wortartmarkierern wiederholt, weshalb eine Alternative nicht existiert und deshalb die komplette manuelle Überprüfung nötig macht.

Zusätzlich wird mit Hilfe von NLTK[BKL09] der dort verfügbare Teil vom WSJ-Korpus ausgegeben. Dieser Korpus ist bereits markiert.

4.2.2. Wortartmarkierungsmenge

Da die Wahl der Menge der Wortartmarkierungen zu unterschiedlichen Goldstandards führt wird als Grundstandard für die Kennzeichnung der Wortartmarkierungen die Penn-Wortartmarkierungsmenge ausgewählt. Sie besteht aus 36 Markierungen für Worte und 12 weiteren Markierungen für Symbole und Zeichen wie Punkte und Währungen. Die Markierungen sind in Tabelle 4.1 aufgelistet.

Die Penn-Wortartmarkierungsmenge wird ausgewählt, da es die am weitesten verbreitete Wortartmarkierungsmenge ist. Die meisten Markierer benutzen die Penn-Wortartmarkierungsmenge oder eine Abwandlung, die sich leicht in diese Wortartmarkierungsmenge übersetzen lässt. Auch benutzt der „American National Corpus“[IS04] neben anderen den Penn-Standard, was erlaubt, einen alternativen Trainingskorpus zu nutzen. In dieser Arbeit werden daher nur Wortartmarkierer genutzt, die ihre Ergebnisse in der Penn-Wortartmarkierungsmenge ausgeben oder sich leicht in diese übersetzen lassen.

4.2.3. Korpora

Für die Performanzanalyse werden ein Teil des WSJ-Korpus und zwei weitere Korpora zur Verfügung gestellt. Diese unterscheiden sich in ihrem Anwendungsbereich und der Formulierung der Sätze. Die Texte der Korpora sind in englischer Sprache.

4.2.3.1. NLCI

Erster Teilkorpus ist der NLCI-Korpus[Ham12], welcher dazu gedacht, ist eine Animation für das Programm Alice [CAB+00] zu beschreiben. Hierbei werden Abläufe verschiedener Akteure aus der Perspektive eines Erzählers beschrieben. Szene, einzelne Akteure und ihre Handlungen werden in der dritten Person in vollständigen Sätzen ausformuliert. Die Rede der Personen ist direkt, die Zeitform für gewöhnlich das „simple present“. Der Korpus gleicht einem Drehbuch und besteht aus 18124 markierten Tokens inklusive 2372 Satzzeichen.

Beispielsätze aus dem NLCI-Korpus

- There is a hole in the ground in the foreground on the right.
- Then the cheerleader turns toward the penguin and the penguin turns its head to the cheerleader.
- Before the dog turns to face the windmill, the dragon said, ”What a beautiful windmill”.
- The bunny turns a bit right to face the mailbox and hops 3 times to the left

Tabelle 4.1.: Das Penn-Tagset nach Marcus et al. [MMS93]

CC	Coordinating conjunction	TO	*to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential *there*	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund or present participle
IN	Preposition or subordinating conjunction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd person singular present
JJR	Adjective, comparative	VBZ	Verb, 3rd person singular present
JJS	Adjective, superlative	WDT	*wh*-determiner
LS	List item marker	WP	*wh*-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	*Wh*-adverb
NNS	Noun, plural	#	Pound sign
NP	Proper noun, singular	$	Dollar sign
NPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

part of the screen, and sits down in front of the mailbox.

4.2.3.2. PARSE

Der zweite Teilkorpus ist der PARSE-Korpus[WT15]. Hierbei spricht ein Proband direkt mit dem Roboter „Armar“ und erteilt ihm Befehle, welche dieser ausführen soll. Die Sätze sind meist kurz und in gesprochener Sprache formuliert, wodurch teilweise grammatikalisch fehlerhafte Sätze enthalten sind. Auch sind Eigenarten der sprachlichen Kommunikation erhalten geblieben, zum Beispiel „uhm“ bei Bedenkpausen und fehlende Satzzeichen jeglicher Art. Der Korpus besteht aus 2174 markierten Worten.

Beispielsatz aus PARSE-Korpus

Armar // I would like to have // uhm // cup of orange juice//
you find the cup on the table // and the juice in the fridge

Die „//“ sind nicht Teil des Korpus und symbolisieren hier die Zeilenumbrüche.

4.2.3.3. WSJ

Der WSJ-Korpus[MMS93] enthält Texte aus dem *Wallstreet Journal*. Die Texte sind wirtschaftlicher Art und befassen sich wirtschaftliche Themen.

Mit 94085 Token ist dieser der größte Korpus.

Beispielsätze aus WSJ-Korpus

- Lorillard Inc. , the unit of New York-based Loews Corp. that makes Kent cigarettes , stopped using crocidolite in its Micronite cigarette filters in 1956 .
- Although preliminary findings were reported more than a year ago , the latest results appear in today 's New England Journal of Medicine , a forum likely to bring new attention to the problem .
- A Lorillard spokewoman said , ” This is an old story .

4.3. Wortartmarkierer

Die Wortartmarkierer und ihre Funktionsweise bilden die Grundlage der darauffolgenden Kombination. Die Auswahl der Wortartmarkierer kann daher große Auswirkungen auf das Endergebnis haben. Um eine möglichst große Bandbreite an Möglichkeiten abzudecken wird eine Vielzahl verschiedener Wortartmarkierer recherchiert. Aus dieser Gesamtmenge werden einige Wortartmarkierer ausgewählt.

Als wichtigste Voraussetzung bei der Auswahl der Markierer wird, da es für die Vergleichbarkeit der Markierer nötig ist, die Verwendung der Penn-Wortartmarkierungsmenge festgelegt. Ein maschinelles Übersetzen zwischen Wortartmarkierungsmengen ist aufgrund von Mehrdeutigkeiten nur schwer möglich, weshalb manuelle Nachbearbeitung fast immer nötig ist. Um diesen Aufwand zu verhindern muss jeder Wortartmarkierer in der Penn-Wortartmarkierungsmenge markieren können. Durch Festlegen der Wortartmarkierungsmenge als Voraussetzung wird die Sprache auf Englisch festgelegt.

4.3.1. Ausgewählte Wortartmarkierer

Für die Auswahl der zur Weiterverwendung genutzten Wortartmarkierer wird darauf Wert gelegt, dass die einzelnen Markierer möglichst unterschiedliche Algorithmen zur Markierung der Worte implementieren. Nutzen verschiedene Markierer den gleichen oder einen

Tabelle 4.3.: Übersicht über die ausgewählten Wortartmarkierer

Wortart-markierer	Trainings-korpus	Kurzbeschreibung	Wortge-nauigkeit
Jitar	WSJ	HMM	K.A.[1]
Berkley Parser	WSJ	Grammit mit Abhängigkeitsbäumen	K.A.
NLTK Brill	WSJ	Linguistische Regeln	K.A.
MBSP	WSJ	Instanzbasierte Methode	97,70%
MATE	WSJ	MaxEnt mit bidirektionalen zyklischen Abhängigkeitsnetzen	97,49%
clearNLP	WSJ	Dynamische Modelauswahl	97,46%
Lookahead POS Tagger	WSJ	Perceptron mit Voraussicht (engl. Lookahead)	97,33%
LTAG-spinal	WSJ	Bidirektionale Suche	97,33%
SENNA	K.A.	Vereinigte Neurale Netzwerkarchitektur	97,29%
GPOSTTL	K.A.	Brill mit eigener Lemma- und Tokenliste	97,24%
Stanford	WSJ	MaxEnt mit bidirektionalen zyklischen Dependenznetzen	97,24%
SVMTool	WSJ	Stützvektormaschinensuche	97,16%
Illinois POS Tagger	WSJ	Netz aus linearen Klassifikatoren	97,13%
crfTagger	WSJ	Konditionale Zufallsfelder	97,00%
PyGreedyAP	WSJ	Perceptron	96,80%
OpenNLP MaxEnt	Leipzig	MaxEnt mit bidirektionalen zyklischen Abhängigkeitsnetzen	96,59%
TreeTagger	WSJ	Kombiniert HMM und Entscheidungsbäume	96,36%

sehr ähnlichen Algorithmus, dann wurde nur ein Vertreter ausgewählt und die anderen verworfen.

In Tabelle 4.3 sind die genutzten Wortartmarkierer mit der in der Literatur angegebenen Genauigkeit und einer Kurzbeschreibung aufgelistet.

4.3.1.1. Stanford Log-linear POS-Tagger

Der *Log-linear POS-Tagger* der *Natural Language Processing Group* aus Stanford ist ein dem aktuellen Stand der Technik entsprechender Wortartmarkierer[TKMS03]. Er basiert auf probabilistischen Grundlagen und verwendet zyklische Dependenznetze, um lokale Merkmale zu verbinden. Der Markierer kann daher, anders als viele andere Wortartmarkierer, die nur unidirektional klassifizieren können, Worte durch bidirektionale Information markieren.

Eine Eigenschaft des Stanford-Markierers ist, dass er oft in wissenschaftlichen Arbeiten als Vergleich für andere Wortartmarkierer verwendet wird. Zum Trainieren des Markierers werden die Abschnitte 0 bis 18 des WSJ-Korpus des „*Penn Treebank*“-Projekts verwendet. Die Abschnitte 19 bis 21 dienen dazu, das Modell zu optimieren; auf den Abschnitten 22 bis 24 findet die Evaluation statt.

[1] bedeutet „In der Literatur konnte keine Angabe gefunden werden.“

4.3.1.2. OpenNLP

OpenNLP[Apa15] ist ein Softwarewerkzeugkasten für die Verarbeitung natürlicher Sprache und liefert zwei Wortartmarkierer.

Der erste Wortartmarkierer ist ein Maximum-Entropie-Markierer, der von links nach rechts Worte untersucht und diese dann nach Wortmerkmalen klassifiziert.

Ein Perceptron-Markierer bildet den zweiten Wortartmarkierer des *OpenNLP*.

4.3.1.3. ClearNLP

Grundidee des *ClearNLP-Markierers* von Jinho Choi und Martha Palmer[CP12] ist, den Effekt der Überanpassung eines Markierers an seinen Trainingskorpus auszunutzen. Dabei werden auf demselben Korpus zwei Modelle trainiert, ein allgemeines und ein spezifisches.

Zum Markieren eines Satzes wird aus diesen beiden Modellen jenes ausgewählt, das dem Satz am ähnlichsten ist.

Wegen des Fokus auf die Zweimodellarchitektur nur ein simpler Markierer verwendet wird, so erreicht *ClearNLP* auf dem WSJ-Korpus eine Genauigkeit von 97,46% und zählt damit zu den besten Markierern.

4.3.1.4. Illinois POS Tagger

Der *Illinois POS Tagger*[RZ98] der *Cognitve Computation Group* aus Illinois markiert Worte durch ein dreischichtiges Netz linearer Klassifikatoren. Diese linearen Klassifikatoren nutzen Wortmerkmale zur Klassifizierung.

Die einzelnen Schichten bestehen aus einer Eingangsschicht, den einzelnen wortartspezifischen Unternetzen in den verborgenen Schichten und einer Ausgangsschicht, die das beste Ergebnis aus der vorherigen Schicht auswählt. Diese linearen Klassifikatoren der verborgenen Schicht werden alle auf dem gleichen Trainingskorpus separat trainiert.

Beim Markieren werden alle Klassifikatoren gleichzeitig angewendet und das Unternetz zur Klassifikation genutzt, das die höchste Konfidenz besitzt.

Der Wortartmarkierer erreicht auf dem WSJ-Korpus eine Genauigkeit von 97,13%.

4.3.1.5. TreeTagger

Der *TreeTagger* von Helmut Schmid[Sch94] benutzt binäre Entscheidungsbäume, um die Übergangswahrscheinlichkeiten der aktuellen Markierungssequenz zu bestimmen. Diese Entscheidungsbäume überprüfen auf jeder Ebene die Existenz einer vorangehenden Wortart und ersetzen so HMMs. Sie werden rekursiv mit Hilfe von N-Grammen aus einem Trainingskorpus mit möglichst unterschiedlichen Untermengen aufgebaut und danach zurechtgestutzt.

Das Nachschlagen findet über eine rekursive Abfrage der Vorgängermarkierungen statt. Die Übergangswahrscheinlichkeiten für die aktuelle Wortart sind in Blättern gespeichert.

Auf dem WSJ-Korpus erreicht er damit eine Genauigkeit von 96,46%.

4.3.1.6. Jitar

Jitar von Daniël de Kok[de] ist ein in Java implementierter HMM-Wortartmarkierer, der über Trigramme Worte klassifiziert. Er basiert auf der Arbeit von Brants[Bra00] und ähnelt damit dem *TnT-Tagger*.

Der Markierer selber besteht aus drei Teilmarkierern, die jeweils als N-Gramm-HMM implementiert sind.

Während es möglich ist, die drei Teilmarkierer hintereinander zu schalten, werden sie bei *Jitar* durch eine gewichtete lineare Glättung gleichzeitig befragt. Um unbekannte Worte klassifizieren zu können wird eine Suffixanalyse durchgeführt.

Brants gibt für den TnT-Markierer, auf dem Jitar beruht, eine Genauigkeit von 96,7% auf dem WSJ-Korpus an.

4.3.2. Ausgewählte Zerteiler

Neben den Wortartmarkierern gibt es auch Zerteiler, die den Worten ihre Wortarten zuweisen. Obwohl sie für mehr Funktionen als Wortartmarkierung zuständig sind werden sie nur als reine Wortartmarkierer verwendet. Dieser Abschnitt beschreibt die Zerteiler, die implementiert werden.

4.3.2.1. SENNA

SENNA von Ronan Collobert[CWB+11] ist eine allgemeine Architektur, die ein vereinheitlichtes neurales Netz nutzt, um ganze Sätze zu markieren und zeitgleich zu zerteilen. Das neurale Netz baut dabei rekursiv einen Syntaxanalysebaum auf, anhand welchem man die Wortartmarkierungen ablesen kann. Eine Besonderheit ist, dass Merkmale nicht vorher von Hand erzeugt werden, sondern vom neuralen Netz selbständig erlernt werden.

Die Genauigkeit wird hierbei mit 97,29% auf dem WSJ-Korpus angegeben.

4.3.2.2. MATE

MATE von Bernd Bohnet[BN12] ist eine Softwareumgebung zum Analysieren natürlicher Sprache. Die grundlegende Idee ist, dass ein gemeinsamer Ansatz zum Zerteilen und Markieren der Sätze bessere Ergebnisse liefern kann als ein separater Ansatz.

Zum Aufbau dieser Bäume wird ein dem *Lookahead Part-Of-Speech Tagger* von Tsuruoka ähnlicher Algorithmus verwendet. Wie auch bei Tsuruoka nutzt *MATE* einen Kellerspeicher für den aktuellen Baum und eine Warteschlange für den Satz selbst.

Anhand der so aufgebauten Dependenzbäume lassen sich die Wortarten ablesen und eine Genauigkeit von 97,28% erreichen.

4.3.2.3. Berkeley-Parser

Der *Berkeley-Parser* der *Berkeley Natural Language Processing Group*[PBTK06] erstellt eine Grammatik, die dazu genutzt werden kann, Dependenzbäume zu generieren. Die Grammatik zum Ableiten des Dependenzbaums wird dabei aus einer einfachen Grundgrammatik durch Aufspaltung von Symbolen erweitert.

Da eine hohe Komplexität der Grammatik dazu führt, dass der Suchraum schnell sehr groß wird, kann nach dem Aufspalten eine Verschmelzung der Symbole durchgeführt werden. Diese wird dort durchgeführt, wo der kleinste verschmelzungsbedingte Genauigkeitsverlust stattfindet. Die Verschmelzung hat auch den Vorteil, dass Überanpassung reduziert wird.

4.3.3. Nicht Implementierte Markierer

Aufgrund zeitlicher Beschränkung konnten nicht alle ausgewählte Markierer verwendet werden. Bei der Auswahl, welche Wortartmarkierer tatsächlich implementiert werden, wurde auf die Einfachheit der Implementierung geachtet. Es wird daher empfohlen diese Markierer zu späterer Zeit ebenfalls zu verwenden.

4.3.3.1. Brill-Tagger

Der *Brill-Tagger* von Eric Brill[Bri92] ist ein tranformationsbasierter Wortartmarkierer, der anders als die meisten der ausgewählten Markierer nicht propabilistisch markiert, und damit den regelbasierten Markierern zuzuschreiben ist.

Eric Brill gibt eine Genauigkeit von 94,9% auf dem Brown-Korpus[KF67], bei Verwendung von 71 Regeln, an.

4.3.3.2. crfTagger

Der *crfTagger* von Xian-Hieu Phan[Pha06] benutzt zum Klassifizieren der Eingabe die namensgebenden Konditionalen Zufallsfelder (engl. Conditional Random Fields (CRF)). CRFs sind probabilistische Modelle, die die Wahrscheinlichkeit einer Sequenz, abhängig von Beobachtungen, berechnen. Anders als HMMs arbeiten sie global mit allen Merkmalen und nicht nur lokal in einem Fenster.[LMP01]

Die angegebene Genauigkeit auf dem WSJ-Korpus beträgt 97,0%.

4.3.3.3. GPoSTTL

GPoSTTL von Golam Mortuza Hossain[Hos06] basiert auf dem *Brill-Tagger*. Er erweitert ihn um einen Tokenisierer, eine Methode, um unbekannte Numerale korrekt zu markieren und fügt dem originalen Lexikon des *Brill-Taggers* Lemmata hinzu.

4.3.3.4. SVMTool

Das *SVMTool* von Jesús Giménez und Lluís Màrquez[GM04] ist ein auf Stützvektormaschinen basierender Wortartmarkierer. Er wird dadurch trainiert, dass für jede Wortart ein eigenen Klassifikator erstellt und für jedes bekannte Wort die Menge an allen möglichen Wortarten in ein Lexikon geschrieben wird. Beim Markieren werden nun Hypothesen generiert und die Wortart mit der höchsten Konfidenz ausgewählt.

Das *SVMTool* erreicht so eine Genauigkeit von 97,2% auf dem WSJ-Korpus.

4.3.3.5. MBSP

Der *Memory Based Shallow Parser* implementiert von Vincent van Asch und Tom de Smedt basierend auf der Arbeit von Walter Daelemans[DZBG96] basiert auf dem Prinzip der Speicherung von Instanzen.

Mit Hilfe dieser gespeicherteren Instanzen wird ein Modell erstellt, das einem unmarkiertem Wort die im Attributraum nächstliegendste Instanz zuweist.

Als Vertreter der instanzbasierten Klassifikatoren erreicht er eine Genauigkeit von 96,4% auf dem WSJ-Korpus.

4.3.4. Nicht verwendete Syntaxanalysierer

Wie auch bei den Wortartmarkierern konnten aus zeitlichen Gründen nicht alle tatsächlich implementiert werden.

4.3.4.1. LTAG-spinal

Der *LTAG-spinal Parser* von Libin Shen et al[SCJ07] ist ein Syntaxanalysierer, der ebenfalls Wortartmarkierungen durchführen kann. Der Fokus seiner Arbeit liegt aber auf der Rückgrat-Variante der Lexikalisierten Baumnachbargrammatiken (engl. „Lexicalized Tree Adjoining Grammar (LTAG)“).

Mit Hilfe von lexikalischen Informationen werden Hypothesen aufgestellt, die anhand einer bidirektionalen Suche zum wahrscheinlichsten Baum führen und anhand welchem die Wortarten abgelesen werden können.

Tabelle 4.4.: Aussortierte Wortartmarkierer

Wortartmarkierer	Ähnlicher Markierer
Smile	Citar
CLL-Tagger	Stanford
MorphAdorner	Citar/GATE
JunkTagger	Stanford
TnT	Citar
CST Brill Tagger	NLTK Brill
LingPipe	Citar
Jitar	Citar
FastTag	NLTK Brill
ACOPost	Diverse
fnTBL	NLTK Brill
muTBL	NLTK Brill
TATOO	Citar
YamCha	SVMTool
MALLET	crfTagger
Sujit Pal's HMM Tagger	Citar
HUMPOS	Citar
CRF++	crfTagger
FreeLing	Citar
QTag	Citar
MXPost	Stanford
topia.termextract	Zu simpel (naiver Ansatz)

4.3.4.2. Lookahead Part-Of-Speech Tagger

Der *Lookahead-Markierer* von Tsuruoka[TMi11] basiert auf der Idee, dass ein unidirektionales Modell, dadurch verbessert werden kann, dass man Informationen aus den nächsten Worten nutzt. Damit soll ein Ergebnis erreicht werden, dass mit global agierenden Markierern konkurrieren kann.

Grundbausteine des Algorithmus sind eine Warteschlange, die den Satz enthält, ein Stapelspeicher, der dazu gedacht ist einen Dependenzbaum zu erzeugen und zwei Operationen. Die Operationen sind *schieben* (lege ein Wort aus der Warteschlange auf den Kellerspeicher) und *reduziere* (vereine zwei Worte mit Dependenz).

Es wird eine Genauigkeit von 97,22% auf dem WSJ-Korpus angegeben.

4.3.5. Abgelehnte Markierer

Während 69 verschiedene Wortartmarkierer gesammelt wurden, waren nicht alle nützlich für die weitere Verwendung. Dieser Abschnitt kategorisiert verschiedene Markierer anhand der Gründe für ihre Ablehnung.

Es wird versucht eine möglichst hohe Diversität an unterschiedlichen Algorithmen zu untersuchen. Implementierten daher verschiedene Markierer einen ähnlichen oder gleichen Algorithmus, dann wurde nur ein einziger Vertreter für die Gruppe ausgewählt. Weiterhin werden Markierer, wie der *topia.termextract*, die nur nach Einzelpunkten wie einer Wortfrequenzanalyse, markierten, ignoriert. In Tabelle 4.4 sind die Markierer und die verwendete Alternative aufgezählt.

Diese Kategorie enthält die Markierer, zu denen zwar nützliche Informationen gefunden wurden, die aber leider aufgrund verschiedener technischer Hindernisse nicht weiter beachtet werden konnten. Die Hauptursachen für die Ablehnung waren oft Gründe wie die

Tabelle 4.5.: Technische Probleme

Wortartmarkierer	Begründung
structRe	Markierer nicht auffindbar
CLAWS4	Falsche Markierungsmenge [Claws4]
TagChunk	Codeformat
NLPdotnet	Zu wenig Informationen
Melt	Falsche Sprache
GENIATagger	Domäne ist Biomedizin
ApacheUIMA	Falsche Markierungsmenge[Brown]
TweetNLP	Domäne ist Twitterkurznachrichten
TagMiner	Markierer nicht auffindbar
Xerox Online Tagger	Zu wenig Informationen
unsupos	Tokenanzahl zum Trainieren zu gering
LIA_TAGG	Zu wenig Informationen
Sequor	Falsche Sprache
Morfette	Falsche Sprache
LinguaStream	Falsche Sprache

falsche Wortartmarkierungsmenge oder das eine lauffähige Implementierung des Wortartmarkierers nicht auffindbar war. Eine Begründung für die Ablehnung der einzelnen Wortartmarkierer ist in Tabelle 4.5 dargestellt.

4.3.6. Performanzanalyse

Um die Qualität der einzelnen Wortartmarkierer vergleichen zu können, muss eine Performanzanalyse durchgeführt werden.

Die Performanzanalyse der Wortartmarkierer findet durch einen Vergleich der verschiedenen Genauigkeiten statt. Als zu betrachtende Werte werden die Wortgenauigkeiten der einzelnen Wortartmarkierer ausgewählt. Die Genauigkeit wird in einem Tabellenkalkulationsprogramm berechnet. Dabei werden die Wortarten der einzelnen Markierer mit dem Goldstandard verglichen. Die Genauigkeit entspricht dem Anteil der korrekt markierten Worte.

Da nicht alle verwendeten Goldstandards mit dem WSJ-Korpus übereinstimmt, wird jeder Wortartmarkierer zunächst einzeln geprüft und seine Genauigkeit auf dem verwendeten Korpus festgestellt. Die Genauigkeit auf dem WSJ-Korpus wird mit der vom Entwickler angegebenen Genauigkeit verglichen, falls möglich.

In Tabelle 4.6 sind die Genauigkeit des Vergleiches anhand des NLCI-Korpus sortiert dargestellt. Die Differenz δ der Ergebnisse zwischen NLCI und PARSE ist in Prozentpunkten (%Punkt) dargestellt.

4.3.6.1. WSJ-Korpus

Es ist ersichtlich, dass die Wortartmarkierer auf dem WSJ-Korpus die besten Ergebnisse erzielen. Am besten schneidet hierbei der Jitar-Wortartmarkierer mit einer Genauigkeit von 98,54% ab. Dabei sollte aber beachtet werden, dass die verwendeten Abschnitte des Trainingskorpus der Modelle nicht bekannt sind und deshalb ein Test auf Trainingsdaten stattfinden könnte. Bei Jitar ist dies der Fall, was das überdurchschnittlich gute Ergebnis, durch Überanpassung, erklärt. Es kann angenommen werden, dass dies auf anderen Wortartmarkierer, die auf dem getesteten Teil ein besseres Ergebnis erzielen als angegeben, ebenfalls zutrifft. Als einziger weiterer Markierer erreicht der Stanford-Wortartmarkier mit dem bidirektionalem Modell mit 98,02% eine Genauigkeit von über 98%.

Tabelle 4.6.: Performanzanalyse der Wortartmarkierer

Wortartmarkierer		Genauigkeiten in % WSJ		NLCI	PARSE	δ_{NLCI}^{PARSE}
Name	Modell	Messung	Literatur	Messung	Messung	in %Punkt
Stanford	bidirectional	98,02	97,24	96,17	90,66	-5,50
Berkley Parser		96,94		95,86	86,61	-9,24
SENNA		97.72	97,29	95,85	89,70	-6,15
OpenNLP	MaxEnt	96,71	96,59	95,17	90,11	-5,06
MATE		95,38	97,49	94,91	89,60	-5,30
Stanford	3 words left	97,83		94,45	89,56	-4,89
clearNLP		95,75	97,46	93,94	89,47	-4,93
TreeTagger		92,58	96,36	92,58	80,04	-12,55
OpenNLP	Perceptron	95,98		92,41	85,33	-7,08
Jitar		98,54		91,71	85,37	-6,33
Illinois		91,27	97,13	86,36	83,53	-2,83

4.3.6.2. NLCI-Korpus

Auf dem NLCI-Korpus ist der Wortartmarkierer mit der höchsten Genauigkeit von 96,17% der Stanford-Wortartmarkier mit dem bidirektionalem Modell. Das zweitbeste Ergebnis hat der Berkeley Parser mit einer Genauigkeit von 95,86%.

4.3.6.3. PARSE-Korpus

Für den PARSE-Korpus ist ersichtlich, dass die Wortartmarkierer generell schlechter abschneiden als auf dem NLCI-Korpus. Da der PARSE-Korpus auf gesprochener Sprache basiert, ist die Distanz zum WSJ-Korpus, der meist zum trainieren verwendet wurde, größer. Dies führt zu generell schlechteren Ergebnissen. Der beste Wortartmarkierer ist auch hier der Stanford-Wortartmarkierer mit dem bidirektionalen Modell mit einer Genauigkeit von nur 90,66%.

4.3.6.4. Fazit

Durch die Performanzanalyse können die Wortartmarkierer und Korpora mit einander verglichen werden. Dabei wurde festgestellt, dass auf dem WSJ-Korpus im allgemeinen die höchsten Genauigkeiten erreicht werden. Der Wortartmarkierer Jitar erreicht auf diesem Korpus das beste Ergebnis mit 98,54% und wird für den WSJ-Korpus als Vergleich genutzt. Auf dem NLCI und PARSE-Korpus konnte sich der Wortartmarkierer von Stanford mit dem bidirektionalen Modell durchsetzen. Er erreichte auf dem NLCI-Korpus eine Genauigkeit von 96,17% und auf dem PARSE-Korpus 90,66%. Beide Korpora nutzen daher den Stanford-Markierer als Vergleichsmarkierer. Das Ergebnis der besten Wortartmarkierer spiegelt sich auch auf den Rest der Markierer wieder. Während die Genauigkeiten der restlichen Markierer auf dem NLCI-Korpus ungefähr bei 95% sind, können nur zwei Markierer die 90% Hürde auf dem PARSE-Korpus überwinden. Die Markierer schneiden damit auf PARSE am schlechtesten ab.

Vergleicht man die Genauigkeiten auf den beiden Korpora, so kann man feststellen, dass bestimmte Wortartmarkierer schlechter abschneiden als der Durchschnitt. Es ist davon auszugehen, dass die Techniken, die von diesen Wortartmarkierern genutzt werden, für gesprochene Texte weniger geeignet sind. So ist beim Illinois-Wortartmarkierer die Differenz zwischen dem NLCI-Korpus und dem PARSE-Korpus am kleinsten. Die dort verwendeten Verfahren sind also am robustesten, leider aber schneidet dieser Markierer unabhängig vom Korpus unbefriedigend ab.

Tabelle 4.7.: Beispielsatz auf dem NLCI Korpus

	Wortartmarkierer		Wortart
Wort	Stanford-bidir	Berkeley Parser	Korrekt
there	EX	EX	EX
is	VBZ	VBZ	VBZ
a	DT	DT	DT
Bunny	NNP	NN	NNP
facing	VBG	VBG	VBG
east	JJ	RB	RB

Die Genauigkeiten der Wortartmarkierer sind auf dem gleichen Korpus unterschiedlich. Dies deutet darauf hin, dass sie von einander unabhängige Fehler machen. In Tabelle 4.7 ist ein Satz aus dem NLCI-Korpus dargestellt. Neben dem Goldstandard stehen auch die Markierungen der beiden besten Wortartmarkierer bei. Wird bei der Kombinierung nur einem Wortartmarkierer vertraut, so wird immer ein Wort falsch markiert. Bei der Kombinierung könnte beim Wort „Bunny“ das Ergebnis des Stanford-Markierers verwendet werden, während beim Wort „east“ auf den Berkeley Parser vertraut wird. Damit wäre der Satz vollständig korrekt markiert.

Insgesamt werden 788 Worte von beiden Markierern mit unterschiedlichen Wortarten versehen.

Bei allen drei Korpora ist daher Raum für Verbesserung. Dabei wirken kleinere Verbesserungen auf Korpora mit hoher Basisgenauigkeit, wie dem WSJ-Korpus, stärker aus als auf Korpora mit niedriger Basisgenauigkeit, wie dem PARSE-Korpus. Wird das Ergebnis von Jitar auf dem WSJ-Korpus durch Kombination um 0,50% verbessert, so führt das dazu, dass die Anzahl Fehler um ein Drittel fällt. Eine Erhöhung von 0,50% auf einem Korpus der eine Basisgenauigkeit von 96,00% hat, würde hingegen nur eine Fehlerreduktion von einem Achtel bedeuten. Wird hingegen eine Erhöhung auf 99,00% erreicht, so wurden 75,00% aller Fehler beseitigt. Vorausgesetzt die Fehler treten alleine auf und führen zu keinen Folgefehlern, so kann dies anhand der Fehlerrate der Sätze verdeutlicht werden. Wenn die durchschnittliche Anzahl Worte pro Satz bei 11 (entspricht ungefähr dem NLCI-Korpus) liegt, würde eine Genauigkeit von 98,50% dazu führen, dass durchschnittlich jeder sechste Satz fehlerhaft ist. Die 96,00% Genauigkeit des NLCI-Korpus würde bedeuten, dass nur der halbe Korpus satzweise komplett korrekt klassifiziert wurde. Eine Genauigkeit von 90,00%, wie sie auf dem PARSE-Korpus erreicht wurde, führt dazu, dass durchschnittlich in jedem Satz mindestens ein Fehler ist. Die Verbesserung der Genauigkeit auf 99,00% führt dazu, dass nur in jedem neunten Satz ein Fehler vorkommt.

4.4. Entwurf

Aus dem in der Einleitung formulierten Problemstellung, leitet sich das Hauptziel dieser Arbeit, die „Optimierung von Wortartmarkierern durch Kombination dieser mit verschiedener Klassifikationsalgorithmen“ ab. Die beiden Teilprobleme führen dabei zu zwei unabhängigen Teilzielen. Die Teilziele erfüllen gemeinsam das Hauptziel.

Das erste Teilziel beschäftigt sich mit der Anbindung und Koordination der Wortartmarkierer. Dies ist eine nötige Vorarbeit, um die Kombinierung durchführen zu können. Eine genauere Analyse findet sich in Unterabschnitt 4.4.3. Hierbei muss beachtet werden, dass die Ausgabe der Wortartmarkierer als Eingabe für die Kombinierung dienen soll. Eine Anforderung ist daher folglich, dass die Ergebnisausgabe vom zweiten Teil möglichst direkt verwendet werden kann.

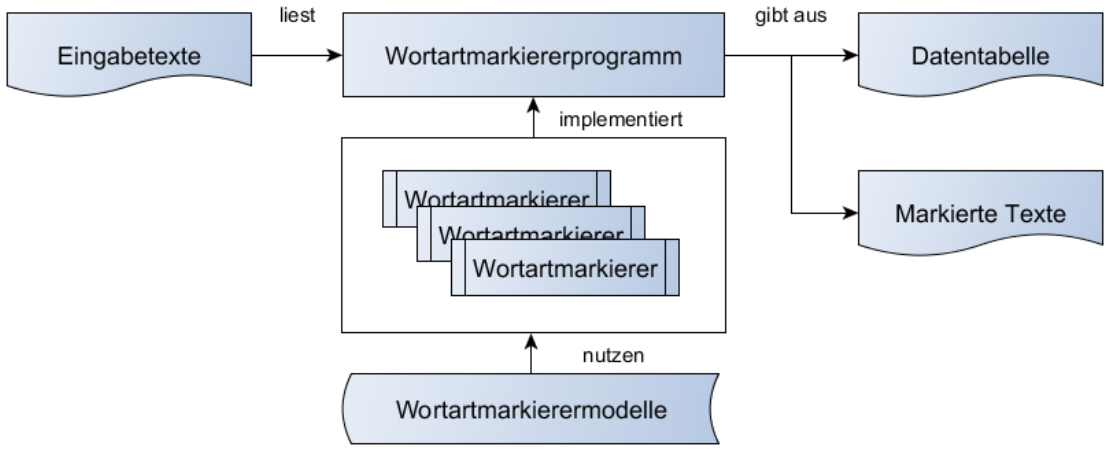

Abbildung 4.1.: Programmablauf für Koordination der Wortartmarkierer

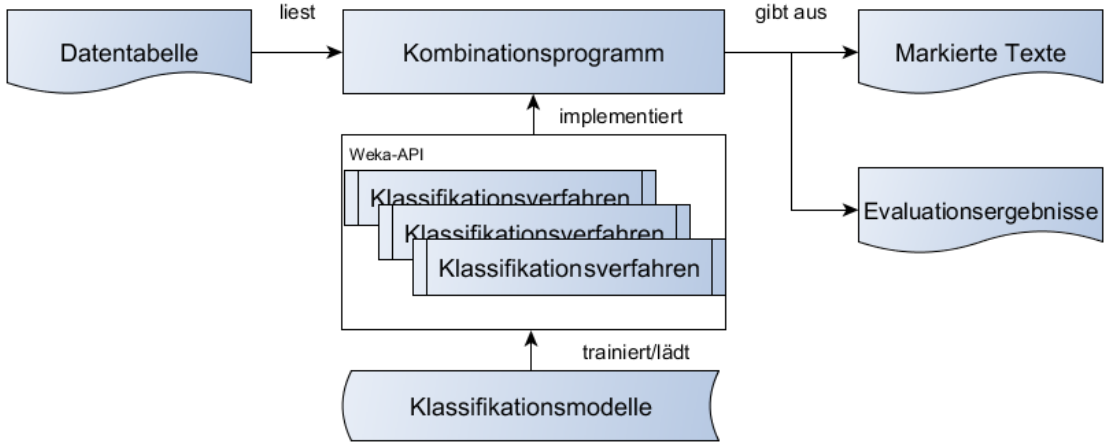

Abbildung 4.2.: Programmablauf für Kombination durch Klassifikationsverfahren

Das andere Teilziel ist die Kombinierung der Ergebnisse der verschiedenen Wortartmarkierer zu einem einzelnen Wortartmarkierer. Dies wird in Unterabschnitt 4.4.4 besprochen.

4.4.1. Genereller Aufbau und Struktur

Zur Erfüllung der definierten Teilziele sollen zwei Programme implementiert werden. Während das erste Programm nur das Wortartmarkierungsziel erfüllen soll, werden die beiden Teilziele der Kombination und Evaluierung dieser Kombinationen von einem einzelnen zweiten Programm erfüllt. Dabei sollen die Ausgabedaten des ersten Programms als Eingabedaten für das zweite Programm dienen.

Das in Abbildung 4.1 dargestellte Wortartmarkierprogramm bekommt Texte und nutzt Wortartmarkierer mit vortrainierten Modellen, um diese zu markieren. Die Ausgabe besteht einmal aus den markierten Texten der einzelnen Markierer und einem Trainingskorpus für das in Abbildung 4.2 abgebildete Kombinationsprogramm. Dieses nutzt verschiedene Klassifikationsverfahren, um auf dem Trainingskorpus ein Modell zu trainieren und zu evaluieren. Die Ausgabe sind wieder mit der Penn-Wortartmarkierungsmenge markierte Texte. Die von den Wortartmarkierern markierten Texte werden in verschiedenen Formaten ausgegeben. Eins dieser Ausgabeformate wird im Kombinationteil als Eingabe genutzt.

Als nicht funktionale Anforderung gilt hierbei die Erweiterbarkeit der einzelnen Teilkomponenten. In den Bereichen maschinelles Lernen und Wortartmarkierung gibt es ständig neue Entwicklungen. Ebenso gibt es bereits eine große Anzahl an bereits verfügbaren Wortartmarkierern und Klassifikationsverfahren. Da von Anfang an klar ist, dass eine große Anzahl an Wortartmarkierern und Klassifikationsverfahren implementiert werden muss, soll das Hinzufügen aktueller und neuer Wortartmarkierer und anderer Module leicht von statten gehen. Durch einen modularen Aufbau wird die einfache Erweiterbarkeit sichergestellt.

4.4.2. Formate

Für die Ein- und Ausgabedaten der Programme werden folgende Dateiformate genutzt:

- Das `txt`-Textformat dient als Eingabe für die Texte. Worte werden, falls nötig, von einer Goldstandardmarkierung getrennt und eingelesen. Weiterhin sollen die Ergebnisse der Wortartmarkierungen, der beiden Teilbereiche, separat in diesem Format ausgegeben werden.
- Das `csv`-Tabellenformat dient als Ausgabeformat für den Teilbereich der Wortartmarkierer. Es stellt nicht nur die Ergebnisse der einzelnen Wortartmarkierer gemeinsam dar, sondern wird auch vom Kombinationsteilbereich als Eingabeformat genutzt. Die Genauigkeit bei der Evaluation der einzelnen Klassifikationsmodelle wird ebenfalls tabellarisch dargestellt.

Der modulare Aufbau der Programme erlaubt es weitere Formate hinzuzufügen.

4.4.3. Wortartmarkierung

Aus der Analyse der Problemstellung folgt das Teilproblem der Organisation der Wortartmarkierer. Dieses Teilproblem betrifft nur die Wortartmarkierer und endet bei der Ausgabe der Ergebnisse dieser.

Um einen Text mit Wortarten zu versehen, müssen folgende Arbeitsschritte durchgeführt werden:

1. Damit ein Text markiert werden kann, müssen die Texte zuerst eingelesen werden.
2. Nach dem Einlesen der Texte müssen die Worte des Textes in Tokens umgewandelt werden.
3. Der tokenisierte Text muss Satz für Satz mit der Wortart markiert werden.
4. Nachdem der Text vollständig markiert ist, kann das Ergebnis ausgegeben werden.

Jeder dieser Arbeitsschritte wird von allen Wortartmarkierern wiederholt. Dabei treten bei jedem dieser Schritte Teilprobleme auf, die beachten werden müssen. So ist es beim Einlesen wichtig auf das korrekte Format zu achten. Mögliche Formate wären ein einfacher Fließtext oder bereits tokenisierte Worte in Zeilen stehend. Deshalb sollte hier die Möglichkeit bestehen, leicht weitere Formate hinzufügen zu können.

Beim Tokenisieren findet auch eine erste Vorverarbeitung des Textes statt. Hier werden zum Beispiel Anführungszeichen in anfangende und abschließende Anführungszeichen aufgetrennt. Dies ist nötig, damit alle Wortartmarkierer die gleichen Tokens nutzen.

Da der verwendete Goldstandard bereits tokenisiert ist, soll das Einlesen und Tokenisieren vom gleichen Modul durchgeführt werden. Dieses Modul soll auch dafür zuständig sein die verwendete Dateistruktur aufzubauen und somit zu standardisieren. Aufgrund der verschiedenen Formate soll es möglich sein leicht weitere Tokenisierer hinzufügen zu können.

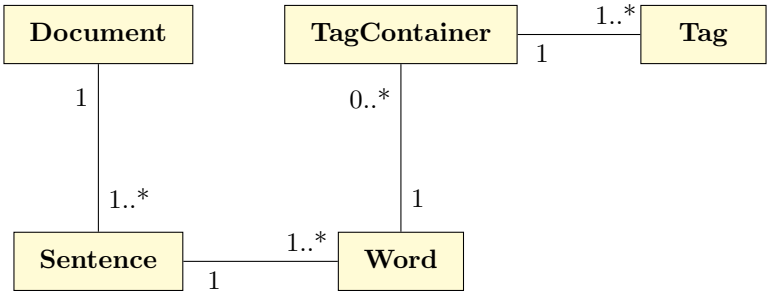

Abbildung 4.3.: `Document`-Strukturentwurf

Beim Markieren besteht das Problem, dass es kein standardisiertes Eingabeformat gibt. Deshalb müssen die Texte für jeden Wortartmarkierer in ein entsprechendes Eingabeformat überführt werden. Weil es ebenfalls keine standardisierte Ausgabe gibt, müssen bei jedem Wortartmarkierer die Ergebnisse in ein einheitliches Format umgewandelt werden. Dies ermöglicht eine leichte Vergleichbarkeit der Ergebnisse. Auch hier sollten, da zum einen viele Wortartmarkierer verwendet werden und zum anderen Neuentwicklungen stattfinden, leicht neue Wortartmarkierer hinzugefügt werden können. Jeder Wortartmarkierer kann weiterhin noch eigene Besonderheiten haben, die selbstständig gelöst werden, ohne den Rest des Programms zu beeinflussen.

Bei der Ausgabe muss beachtet werden, dass eine Ausgabe in einem zur Kombination weiterverwendbarem Format stattfindet. Da die Ausgabe aber nicht nur zur Kombination weiterverwendet wird, sondern auch andere Verwendungszwecke denkbar sind, soll sie in verschiedenen Formaten möglich sein.

Der Vergleich zwischen den Wortartmarkierern kann sehr einfach in Tabellenkalkulationsprogrammen durchgeführt werden, deshalb soll eine Ausgabe in ein passendes Format möglich sein.

Die Wortartmarkierer können unabhängig von einander arbeiten und müssen nur bei der Ausgabe ein gemeinsames Ergebnis präsentieren.

4.4.3.1. Interne Repräsentation

Um eingelesene Texte im Programm nutzen zu können, wird eine Datenstruktur benötigt. Eine Dokument-Datenstruktur soll die einzelnen Worte und Sätze im Wortartmarkierungsteil verwalten. Beim Markierungsschritt entstehen weitere Daten, wie die Markierung und der Name der Markierer, die ebenfalls in der Dokument-Datenstruktur gespeichert werden sollten.

In Unterunterabschnitt 4.4.3.1 ist die Struktur und in Abbildung 4.4.3.1 sind die einzelnen Klassen der Dokument-Datenstruktur (`Document`) dargestellt. `Document` speichert Metainformationen über die Datei und besteht ansonsten nur aus einer Liste von Sätzen (`Sentence`). `Sentence` speichert Worte (`Word`) in einer doppelt verketteten Liste. Damit soll sichergestellt werden, dass leicht auf den Kontext eines Wortes zugegriffen werden kann. `Word` selber besteht aus dem Wort und einer Liste von Container-Objekten (`TagContainer`), die eine Liste von möglichen Wortartmarkierungen (`Tag`) für dieses Wort und den Namen des benutzen Wortartmarkierers enthält. `Tag` enthält die Wortartmarkierung und eine zu dieser Markierung gehörige Konfidenz.

4.4.3.2. Struktureller Aufbau

In Unterunterabschnitt 4.4.3.2 ist der strukturelle Entwurf des Wortartmarkierungsteils dargestellt. Die Hauptklasse bildet der in Abbildung 4.4.3.2 gezeigte `TaggingManager`. Die-

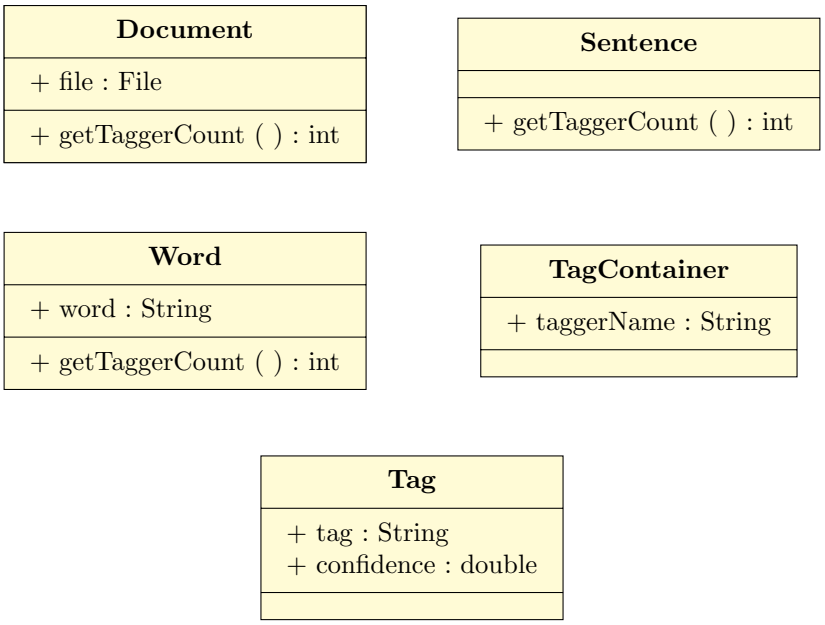

Abbildung 4.4.: Übersicht über die Klassen der `Document`-Datenstruktur.

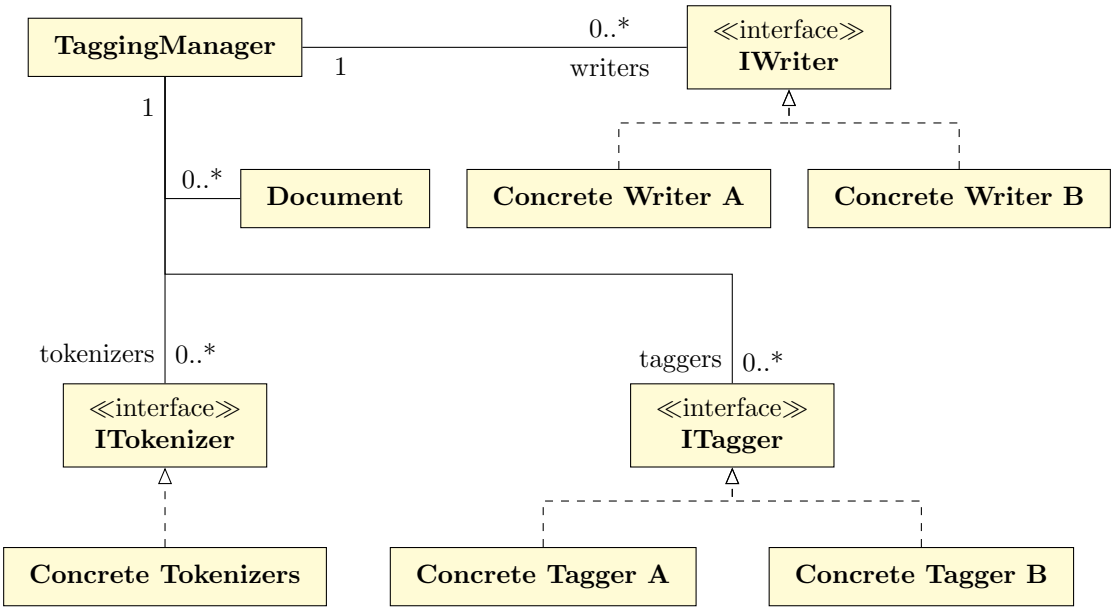

Abbildung 4.5.: `TaggingManager`-Strukturentwurf

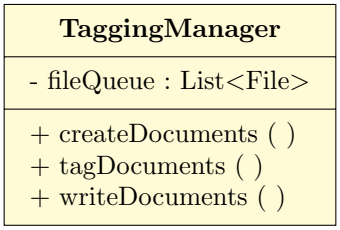

Abbildung 4.6.: `TaggingManager`-Klassenentwurf

ser ist dafür zuständig den Ablauf des Programms und der einzelnen Module zu steuern. Die verschiedenen Wortartmarkierer werden als Module geladen und in einer Warteschlange verwaltet. Das Programm liest über einen geladen *`ITokenizer`* Texte ein und erstellt daraus `Document`-Datenstrukturen. Diese Dokumente werden in einer Warteschlange verwaltet. Die als Module geladenen Wortartmarkierer *`ITagger`* markieren die einzelnen Dokumente der Warteschlange.Die so markierten Texte werden als komplette Warteschlange an die vom `ServiceLoader` geladenen *`IWriter`* übergeben, die sich um eine entsprechende Dateiausgabe kümmern.

4.4.4. Kombinierung der Wortartmarkierer

Das Teilproblem der Kombination der Wortartmarkierer lässt sich in mehrere weitere Teilprobleme zerlegen. Klassifikationsverfahren nutzen Modelle, um Daten zu klassifizieren. Daraus folgt, dass es möglich sein muss, aus Trainingsdaten ein Modell zu erzeugen. Wurde ein Modell erzeugt, so muss dieses Modell geladen werden können. Das geladene Modell kann daraufhin dazu genutzt werden, um Daten zu klassifizieren. Es ergeben sich damit mindestens zwei Betriebsmodi: Training und Klassifikation. Beide Betriebsmodi haben einige Gemeinsamkeiten und Unterschiede. So müssen bei beiden die benötigten Daten zuallererst eingelesen werden. Hierbei ist zu beachten, dass das Ausgabeformat des Wortartmarkiererprogramms direkt als Eingabeformat verwendet werden soll. Die eingelesenen Daten können dann durch Filter vorverarbeitet werden. Nun divergiert der Ablauf abhängig vom Betriebsmodi.

Um aus den Daten ein Modell zu erstellen, werden die eingelesenen Daten als Trainingsdaten für die Klassifikationsverfahren genutzt. Das so erstellte Modell wird für die spätere Verwendung gespeichert.

Zur Klassifikation eines Datensatzes wird ein vorher erstelltes Modell geladen. Anhand dieses Modells werden die eingelesenen Daten klassifiziert. Das Ergebnis der Klassifikation muss in geeigneter Form gespeichert werden.

Weiterhin muss die Qualität der Klassifikationsverfahren evaluiert werden. Die Evaluation wird deshalb als dritter Betriebsmodus implementiert. Damit kann gleichzeitig das ganze Verfahren evaluiert werden, da das beste Klassifikationsergebnis auch das beste durch das Verfahren zu erreichende Ergebnis darstellt. Dies wird mit den in den Grundlagen erklärten Kreuzvalidierungsverfahren erreicht. Dabei soll nicht nur eine 10-fache Kreuzvalidierung durchgeführt werden, sondern auch die Auswirkungen unterschiedlicher Anteilverhältnisse von Test- und Trainingskorpus untersucht werden. Dadurch kann analysiert werden, ob eine Verringerung des benötigten Trainingskorpus möglich ist. Auch erlaubt es eine Aussage darüber, ob bestimmte Klassifikationsverfahren sich besser für einen kleinen Trainingsdatensatz eignen. Das Ergebnis der Evaluation wird dann dem Benutzer zur Verfügung gestellt. Da keine Weiterverarbeitung dieser Daten stattfindet, muss hierbei nur die Lesbarkeit beachtet werden.

Zur Evaluation der einzelnen Klassifikationsverfahren werden die in Abschnitt 2.4 erklärten Metriken verwendet. Im einzelnen sind dies der Erwartungswert, die Varianz und Cohens

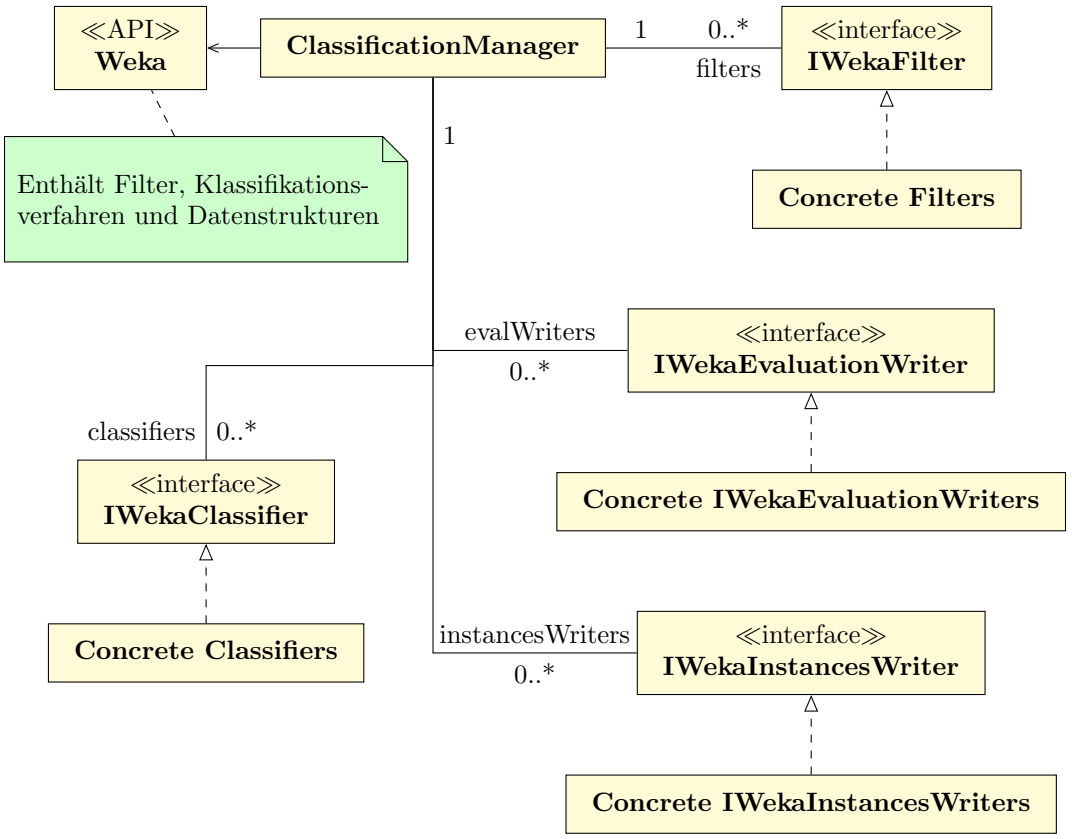

Abbildung 4.7.: `ClassificationManager`-Strukturentwurf

Kappa. Der Erwartungswert entspricht der Genauigkeit und lässt sich daher direkt mit dem besten Wortartmarkierer vergleichen. Da die Genauigkeit von der Zusammensetzung der Trainings- und Testdaten abhängt, wird auch die Varianz, als Abschätzung der Abweichung vom Erwartungswert benötigt. Cohens Kappa gibt Auskunft über die Objektivität der Kreuzvalidierung.

Evaluiert werden die in Abschnitt 2.2 vorgestellten Klassifikationsverfahren. Da eine Vielzahl an Klassifikationsverfahren betrachtet und potentiell weitere Verfahren entwickelt erden könnten, sollte auch hier eine leichte Erweiterbarkeit sichergestellt werden.

4.4.4.1. Interne Repräsentation

Für das Kombinationsprogramm werden verschiedene Datenstrukturen benötigt. Die erste Datenstruktur dient dazu, die eingelesenen Daten zu verwalten. Dies entspricht einer Liste von Instanz-Objekten, die ihre dazugehörigen Attribute verwalten.

Um die Ergebnisse mit mehreren Ausgabeschreibern ausgeben zu können, müssen die Ergebnisse der Evaluation ebenfalls verwaltet werden. Dies ist ebenfalls eine Liste von Evaluation-Objekten, die alle geforderten Metriken der Evaluation gespeichert haben.

4.4.4.2. Struktureller Aufbau und genereller Ablauf

In Unterunterabschnitt 4.4.4.2 ist der strukturelle Aufbau des Kombinationsteils dargestellt. Der in Abbildung 4.4.4.2 dargestellte `ClassificationManager` dient hierbei als Verwaltungseinheit für die einzelnen Module. Die einzelnen Module werden auch hier in Warteschlangen verwaltet.

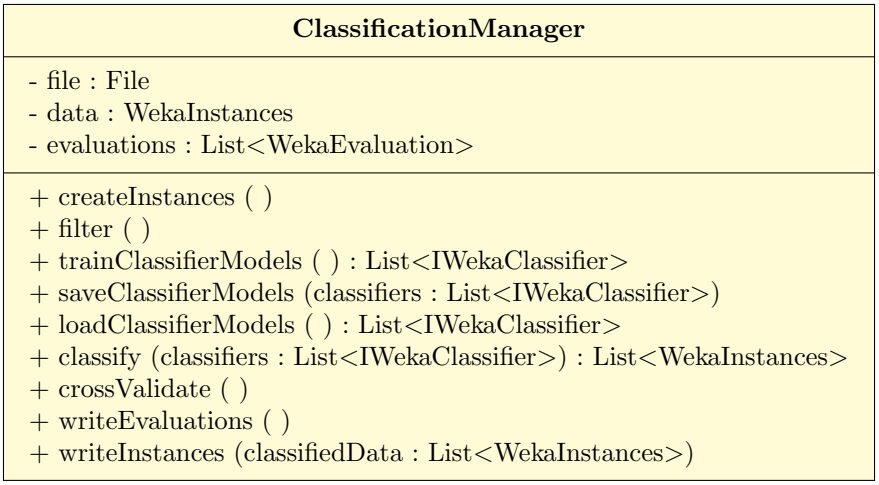

Abbildung 4.8.: `ClassificationManager`-Klasse

`ClassificationManager` ist für die Koordinierung der drei Betriebsmodi und damit auch des Programmes zuständig. Die Auswahl des Betriebsmodus und Übergabe der Eingabedatei findet über die Argumente des Programmes statt. Wurde der Betriebsmodus festgestellt, kann der Programmablauf starten. Der `ClassificationManager` liest eine Textdatei ein, in der die einzelnen Ergebnisse der Wortartmarkierer stehen. Die eingelesenen Daten werden in Instanz-Objekt übersetzt. Dabei wird auch abhängig vom entsprechenden Argument, das Attribut festgelegt, nach welchem klassifiziert wird. Das `Instance`-Objekt wird zuerst von den geladenen *`IWekaFilter`* und *`IWekaClassifier`*-Modulen bearbeitet. Abhängig vom Betriebsmodus bearbeiten die *`IWekaClassifier`*-Modulen die Daten folgendermaßen:

- Modelltraining: Die Module trainieren anhand der eingelesenen Daten Modelle und speichern diese.
- Klassifizierung: Die Module laden das vorher trainierte Modell und klassifizieren die Daten.
- Kreuzvalidierung: Auf den Daten findet eine Kreuzvalidierung statt.

Die Rückgabewerte der *`IWekaClassifier`*-Objekte werden an ihre entsprechenden *`IWekaEvaluationWriter`* oder *`IWekaInstancesWriter`*-Module weitergegeben, die diese in eine entsprechende Datei schreiben.

4.5. Zusammenfassung

In diesem Kapitel wurden die verschiedenen Probleme und Ziele die sich aus der Aufgabenstellung ableiten definiert und analysiert. Um die Problemstellung zu lösen mussten erst einige Herausforderungen gemeistert werden, bevor ein Softwaretechnischer Entwurf angefertigt werden könnte. Als erstes wurden entsprechende Wortartmarkierer recherchiert und daraus eine passende Untermenge zu Verwendung ausgewählt. Weiterhin wurden drei Korpora angefertigt. Auf diesen Korpora wurde eine Performanzanalyse durchgeführt. Dabei konnte bestätigt werden, dass Wortartmarkierer auf Korpora, die eine sprachlich thematische Nähe zum Traininskorpus besitzen, besser abschneiden als solche denen diese Nähe fehlt. Aufgrund dieser Performanzanalyse wurden die besten Wortartmarkierer für eine spätere Evaluation ausgewählt. Die Analyse des Problems führte zum Entwurf einer softwaretechnischen Lösung. Der Entwurf teilt sich dabei in zwei von einander unabhängige Programme. Das erste Programm löst das Problem der Koordination verschiedener

Wortartmarkierer. Dabei wird eine Ausgabedatei erstellt, die vom zweiten Programm als Eingabe genutzt wird. Anhand dieser Eingabe werden verschiedene Klassifikationsverfahren genutzt, um die Wortartmarkierung zu optimieren. Die angefertigten Korpora dienen dabei als Trainings- und Testdaten. Die Implementierung dieses Entwurfs folgt im nächsten Kapitel.

5. Implementierung

Einen der ersten Schritte bei der Verarbeitung von natürlicher Sprache stellt die Wortartmarkierung dar. Um Folgefehler zu vermeiden, ist es daher erstrebenswert eine möglichst geringe Fehlerrate zu erhalten. Eine Möglichkeit die Fehlerrate zu verringern, ist die Verwendung eines Anwendungsfeld spezifischen Korpus. Leider ist es nicht immer praktikabel einen optimalen Korpus zu nutzen. Es soll daher eine alternative Möglichkeit untersucht werden die Ergebnisse der Wortartmarkierung zu optimieren. Das Problem der Wortartmarkiereroptimierung wurde in Abschnitt 4.4 analysiert und verschiedene Teilziele, die zur Lösung des Problems führen, festgelegt. Die Teilziele lassen sich unter „Organisierung und Koordinierung von Wortartmarkierern“ und „Optimierung der Ergebnisse durch Kombination mit Klassifikationsverfahren“ in zwei unabhängige Teillösungen bündeln. Um die festgelegten Ziele zu erreichen, wurde eine softwaretechnische Lösung entworfen, die aus zwei unabhängigen Programmen besteht. Die Programme heißen `tagman` und `claman`. `Tagman` organisiert und koordiniert Wortartmarkierer, um damit Texte zu markieren. Die Ausgabe von `tagman` soll als Eingabe für `claman` dienen. `Claman` trainiert dann auf diesen markierten Texten Modelle für Klassifikationsverfahren. Diese Modelle werden von `claman` genutzt, um aus einer Eingabe optimierte Wortmarkierungen zu klassifizieren. Weiterhin findet eine Evaluation der Klassifikationsverfahren durch `claman` statt. Der Entwurf wird in Java implementiert und das gesamte Projekt wird mit Maven verwaltet.

Die nicht funktionale Anforderung der Erweiterbarkeit wird in den Unterprojekten `TaggingManager` und `ClassificationManager` durch eine Vielzahl von Schnittstellen und die Implementierung der Java `ServiceLoader`-Architektur (engl. Facility) erreicht. Die `ServiceLoader`-Architektur nutzt eine Schnittstelle als Argument und initialisiert durch den Standardkonstruktor zur Laufzeit alle Klassen, die diese Schnittstelle implementieren. Zum Auffinden der Klasse sind zusätzliche Metainformationen, die den Klassennamen und die implementierte Schnittstelle beinhalten, notwendig. Sie werden durch `MetaInfServices` von kohsuke[1] automatisch erzeugt. Die Klassen und Metainformationen werden in Modulen gebündelt. Wird das Modul zu einer `jar` compiliert und dem Klassenpfad hinzugefügt, so kann es vom `ServiceLoader` geladen werden. Alternativ kann das Modul als auch eine Abhängigkeit durch Maven implementiert und geladen werden.

Die Probleme und Herausforderungen, die bei der Implementierung des Entwurfes anfallen, werden in diesem Kapitel erläutert. Weiterhin werden die daraus folgenden Änderungen am Entwurf besprochen. Daraus folgend teilt sich das Kapitel in drei Teile auf. Nachdem

[1] Siehe `http://metainf-services.kohsuke.org/`, zuletzt abgerufen am 22.03.2016.

in Abschnitt 5.1 erklärt wird wie das gesamte Projekt aufgebaut ist, behandelt der Rest des Kapitels die beiden Unterprojekte. Dem logischen Aufbau folgend wird zuerst `tagman` in Abschnitt 5.2 und darauf `claman` in Abschnitt 5.3 besprochen.

5.1. Maven-Struktur

In Abbildung 5.1 ist der strukturelle Aufbau des Maven-Gesamtprojektes dargestellt. `Tagman` und `claman` sind hierbei die zwei von einander unabhängig agierenden Programme. Sie werden als Unterprojekte in dem Hauptprojekt `project` gebündelt.

Das `tagman`-Unterprojekt koordiniert die verschiedenen Wortartmarkierer, um Texte zu markieren. Das Hauptsteuerprogramm `TaggingManager` befindet sich im Modul `tagman-app`. `Tagman-app` ist abhängig von `tagman-api`. In `tagman-api` befinden sich die Schnittstellen *`ITokenizer`*, *`ITagger`*, *`IWriter`* und die `Document`-Datenstruktur. `Tagman-api` enthält damit alles nötige, um weitere Module implementieren zu können. `Tagman-tokenizers`, `tagman-taggers` und `tagman-writers` sind daher von `tagman-api` abhängig. Sie sammeln die konkreten Implementierungen der Tokenisierer, Wortartmarkierer und Ausgabeschreiber als Untermodule. `Tagman-bundle` dient dazu alle verwendeten Module zu bündeln, um so das Programm während der Entwicklung leicht ausführen zu können.

Das `claman`-Unterprojekt dient dazu unterschiedliche Klassifikationsalgorithmen zu trainieren, evaluieren und unbekannte Texte mit Hilfe der trainierten Modelle zu klassifizieren. Hierfür nutzt es die von „Waikato Environment for Knowledge Analysis“ (Weka) bereitgestellten Datenstrukturen und Algorithmen. Die Struktur von `claman` ist der von `tagman` sehr ähnlich. So befindet sich das Hauptsteuerprogramm `ClassificationManager` in `claman-app`. `Claman-app` ist ebenfalls von der API `claman-api` abhängig. Die Schnittstellen, die für weitere konkrete Implementierungen benötigt werden, sind in `claman-api` gesammelt. *`IWekaFilter`* wird zur Implementierung von Filtern genutzt. Die Klassifikationsverfahren erben von der abstrakten `AbstractClassifier`-Klasse, welche *`IWekaClassifier`* implementiert. `AbstractClassifier` befindet sich ebenfalls in `claman-api`. Die beiden Schnittstellen für Ausgabeschreiber sind *`IWekaInstanceWriter`* und *`IWekaEvaluationWriter`*. *`IWekaInstanceWriter`* dient hierbei zur Ausgabe von klassifizierten Datensätzen. Die Ergebnisse der Evaluation werden über *`IWekaEvaluationWriter`* ausgegeben. Durch eine Änderung am Entwurf sind auch die Containerklassen `EvaluationContainer` und `InstancesContainer` für `Evaluation` und `Instances` in `claman-api`. Die konkreten Implementierungeen der Filter, Klassifikationsverfahren und Ausgabeschreiber sind in den entsprechenden Untermodulen `claman-filters`, `claman-classifiers` und `claman-writers` gebündelt. `claman-bundle` bündelt auch hier die verwendeten Module. Da zur Evaluation das Programm mehrmals mit unterschiedlichen Argumenten gestartet werden muss, erfüllt `claman-bmbatch` diese Aufgabe.

Nicht im Entwurf enthalten sind die `kiba-tools`. Sie ermöglichen das Überprüfen des Betriebssystems und ein systemunabhängiges Laden von Pfaden zu Wortartmarkierern und Modellen. `ConfigManager` kopiert dafür eine Konfigurationsdatei in das Benutzerverzeichnis. Die Konfigurationsdatei enthält den Pfad zum nötigen Modell oder Verzeichnis.

Da nicht alle Wortartmarkierer in öffentlichen Ablagen (engl. Repository) verfügbar sind, wurde eine lokale Maven-Ablage erstellt. `Tagman` und `claman` nutzen die lokale Ablage, um nötige Bibliotheken zu implementieren.

Um den Benutzer über die Verarbeitung zu informieren, nutzt das gesamte Projekt den slf4j-Aufzeichner[2]. Für die Entgegennahme der Argumente wird die args4j-Bibliothek[3] genutzt.

[2]Siehe `http://www.slf4j.org/`, zuletzt abgerufen am 22.03.2016.
[3]Siehe `http://args4j.kohsuke.org/`, zuletzt abgerufen am 22.03.2016.

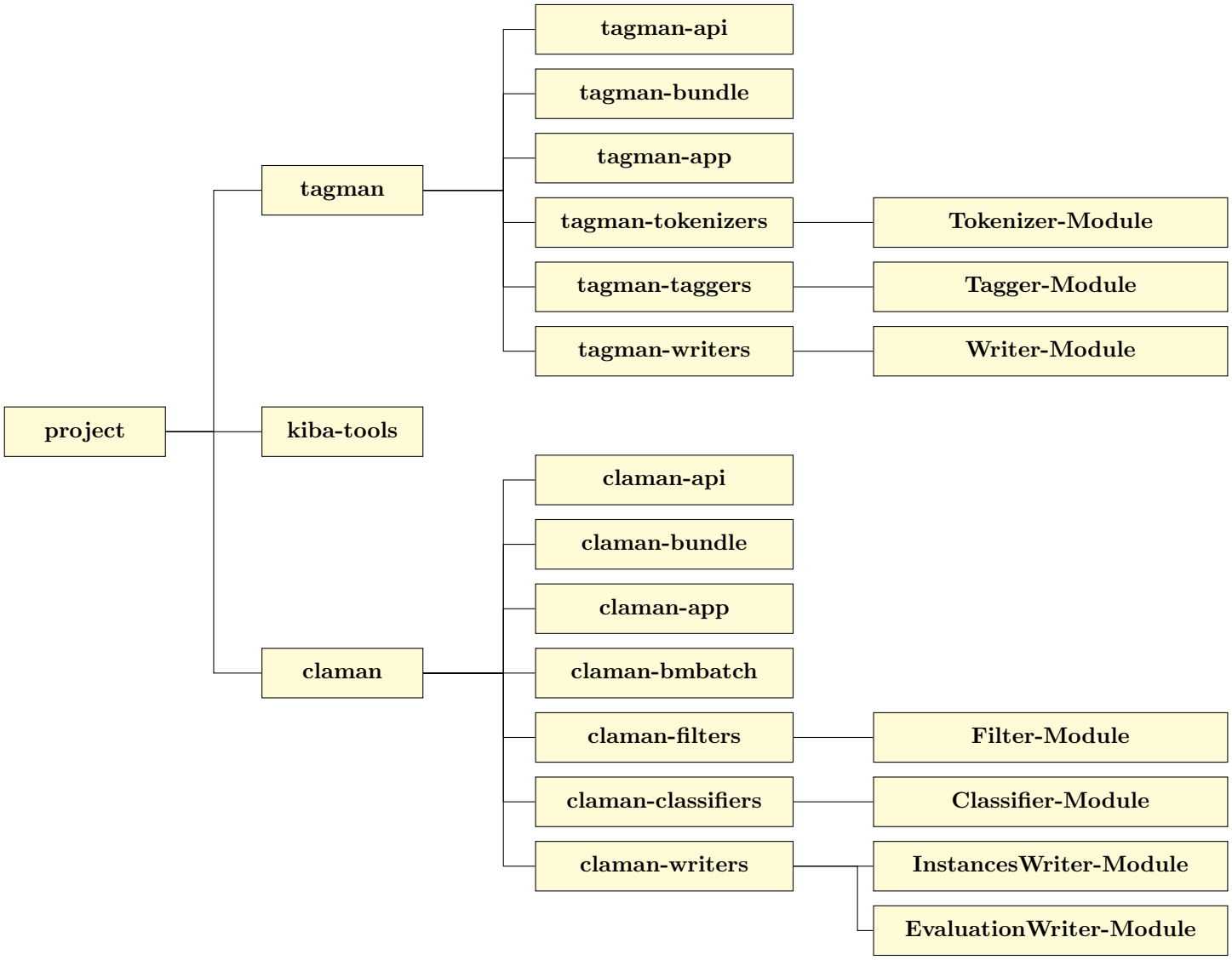

Abbildung 5.1.: Aufbau des Gesamtprojekts

5.2. Tagman-Unterprojekt

Das `tagman`-Unterprojekt ist dafür zuständig Wortartmarkierer zu koordinieren, um so Texte zu markieren. Die Wortartmarkierer sollen gemeinsam Texte markieren und einen Datensatz ausgeben, der von `claman` weiterverwendet werden kann. Der Datensatz enthält Wörter, die korrekte Wortart der Wörter und die Markierung der Wörter durch die Wortartmarkierer. Es soll möglich sein die korrekte Wortart oder die Wörter selber wegzulassen.

In `tagman` nutzen einige Wortartmarkierer die lokale Ablage, um nötige Bibliotheken einzubinden. Die Wortartmarkierer, die die lokale Ablage benötigen, sind der Berkley Parser und der Illinois Pos Tagger.

Außerdem wird `ConfigManager` aus `kiba-tools` benutzt, um Pfade zu Wortartmarkierern und Modellen zu definieren. Das Überprüfen des Betriebssystems, um passende Anwendungen auszuführen, findet mit `OS-Check` aus `kiba-tools` statt. `ConfigManager` wird vom Berkley Parser, Jitar, MATE, OpenNLP, und Senna genutzt. Diese Wortartmarkierer oder Modelle sind daher nicht Teil von `project` und müssen vorher separat installiert werden.

Dies ermöglicht ein systemunabhängigen Programm und damit eine höhere Portabilität.

5.2.1. Document-Datenstruktur

Die `Document`-Datenstruktur wurde, um einige Funktionen erweitert, die es erlauben leichter auf ganzen Sätzen, statt einzelnen Wörtern, zu arbeiten.

Die wichtigsten Funktionen sind `tagSentence(List<TagContainer>)` und `toPlainWordString()`. `TagSentence(List<TagContainer>)` wird dazu verwendet, um alle Wörter eines Satzes mit Hilfe einer `TagContainer`-Liste direkt mit ihrer Wortart zu markieren. `ToPlainWordString()` gibt die Tokens eines Satzes, durch Leerzeichen getrennt, zurück. Der Satz dient in dieser Form als Eingabe für die Wortartmarkierer.

5.2.2. TaggingManager-Hauptklasse

Die `TaggingManager`-Klasse ist die zentrale Steuerklasse des Programmes. Die Klasse ist dafür zuständig, die einzelnen Module zur Laufzeit mit der `ServiceLoader`-Architektur zu laden und zu verwalten.

Die Koordination und Verwendung der Module findet in `TaggingManager` statt. Vor der Markierung werden die Texte geladen, tokenisiert und in `Document`-Objekte umgewandelt. Für den weiteren Ablauf der Hauptschleife gibt es zwei mögliche Lösungen:

1. Alle Wortartmarkierer werden zu Beginn geladen. Ein `Document`-Text wird von allen Wortartmarkierern markiert. Dann wird der nächste `Document`-Text markiert.
2. Ein Wortartmarkierer wird geladen und markiert alle `Document`-Texte durch. Danach wird der nächste Wortartmarkierer geladen und markiert wieder alle Texte durch.

Der Vorteil der zweiten Lösung ist der geringere Arbeitsspeicherverbrauch während des Markierungsschritt. Die Wortartmarkierer müssen dabei nicht gleichzeitig im Speicher gehalten werden sondern können nach dem Markieren verworfen werden. Da geringer Speicherverbrauch nicht Teil der Anforderungen war, wurde die erste Möglichkeit implementiert. Sie ähnelt dem logischen Ablauf und trennt die einzelnen Phasen des Programmes sinnvoll, was eine bessere Lesbarkeit zur Folge hat.

Nachdem alle Wörter der `Document`-Objekte markiert wurden, werden sie von den Schreibern ausgegeben.

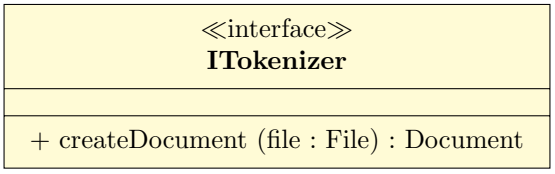

Abbildung 5.2.: `ITokenizer`-Schnittstelle zum Einlesen und Tokenisieren der Textdateien

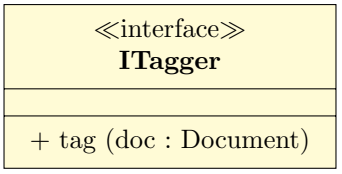

Abbildung 5.3.: `ITagger`-Schnittstelle zur Implementierung der Wortartmarkierer

5.2.3. Modulschnittstellen

Die geforderte Erweiterbarkeit des Programmes führt dazu, dass das Programm modular aufgebaut ist. Alle Module implementieren eine Schnittstelle, die es dem Programmskelett ermöglicht sie anzusprechen. Beim Wortartmarkierungsteil gibt es insgesamt vier Schnittstellen die Module implementieren. Bis auf die Schnittstellen zum Lesen der Eingabedaten und Erzeugen der Dokument-Datenstruktur werden die Modulimplementierungen in einer Warteschlange verwaltet.

Das Einlesen der Texte, der Tokenisierungsschritt und das Erzeugen der Dokument-Datenstruktur findet über die Tokenisierermodule statt. Die Tokenisierermodule werden unter der Schnittstelle *`ITokenizer`* in Abbildung 5.2 gesammelt. Klassen, die diese Schnittstelle implementieren, bekommen über die `createDocument()`-Methode eine Datei, die sie korrekt in die `Document`-Datenstruktur übersetzen müssen.

In Abbildung 5.3 ist die *`ITagger`*-Schnittstelle dargestellt. Unter ihr werden die Module mit den Wortartmarkierern gesammelt. Jedes Modul ist dafür zuständig den entsprechenden `TagContainer` für ein `Document` zu erstellen, über `tag()` zu markieren und wieder in der `Document`-Datenstruktur abzulegen. Die interne Verwaltung der zuständigen Modelle und Einstellungen findet im Modul statt.

Die in Abbildung 5.4 dargestellte *`IWriter`*-Schnittstelle dient dazu die `Document`-Datenstruktur in eine Datei auszugeben. Die Wahl der Datei und der Formatierung wird hierbei der konkreten Implementierung überlassen. Um die Ausgabe von `tagman` als Eingabe von `claman` nutzen zu können, wird das `csv`-Dateiformat verwendet. Es ist hierbei zu beachtet, dass eine Ausgabedatei als Eingabe der Kombination genutzt wird. Weitere Ausgabeformate können für sonstige Zwecke hinzugefügt werden.

5.2.3.1. Tokenisierermodule

Ein Tokenisierer trennt nicht nur Wörter auf, sondern führt einige weitere Änderungen am Text durch. Ein Beispiel für eine solche Änderung wäre die Ersetzung bestimmter Symbo-

Abbildung 5.4.: `IWriter`-Schnittstelle zur Ausgabe der `Document`-Warteschlange

le oder die Unterscheidung zwischen der Position der Anführungszeichen. Unbearbeitete Rohtexte unterscheiden sich daher stark vom Goldstandard. Um ein `Document` zu erzeugen wird die `createDocument(File file)`-Methode der `ITokenizer`-Schnittstelle verwendet. Der Methode wird ein `File` zur Textdatei übergeben, welches in `Document` gespeichert wird. Diese Textdatei wird eingelesen und mit einem Tokenisierer bearbeitet, bevor sie als `Document`-Datenstruktur, zurückgegeben und in `TaggingManager` weiterverarbeitet wird.

Um Texte aus dem Rohformat und dem Goldstandardformat in die `Document`-Datenstruktur einlesen zu können, müssen zwei verschiedene Tokenisierer verwendet werden.

Der `SimpleTextTokenizer` liest bereits tokenisierte Texte im Goldstandardformat ein. Die Wortarten werden hierbei weggelassen und nur die Wörter selber ein die `Document`-Datenstruktur überführt. An der Textstruktur wird hierbei nichts verändert und die Markierungen werden später durch einen speziellen Goldstandardmarkierer eingetragen.

Für Rohtexte wird der `StanfordTokenizer` genutzt. Dieser liest nicht nur die unbearbeiteten Texte ein, sondern tokenisiert sie mit Hilfe des Standford-Tokenisierers, bevor er die tokenisierten Texte in die `Document`-Datenstruktur überführt. Der Tokenisierer von Stanford wird verwendet, weil dieser Tokenisierer auch bei der Erstellung des Goldstandards zum Einsatz kam.

Alle Tokenisierer müssen die Schnittstelle *`ITokenizer`* implementieren, damit sie von der `ServiceLoader`-Architektur geladen werden können.

Es kann immer nur ein Tokenisierer verwendet.

5.2.3.2. Wortartmarkierermodule

Die Koordination der Wortartmarkierer ist eine der Hauptaufgaben dieses Programms. Die einzelnen Wortartmarkierer werden als Module implementiert, die separat geladen und in einer Warteschlange gesammelt werden. Wie im Entwurf festgelegt, werden neue Wortartmarkierer durch Implementierung der `ITagger`-Schnittstelle, hinzugefügt. Im Standardkonstruktor werden die Wortartmarkierer einmalig initialisiert. Alle weiteren vom Text unabhängigen Einstellungen werden hier ebenfalls vorgenommen. In der `tag(Document)`-Funktion findet die komplette Markierung der `Document`-Datenstruktur mit dem Wortartmarkierer statt. Dazu wird den Wortartmarkierer ein `Sentence` in Form eines tokenisierten `String` über `toPlainString()` übergeben. Die Wortarten des Satzes werden vom Markierer in einer `TagContainer`-Liste mit `tagSentence(List<TagContainer>)` dem `Document` übergeben. Ein jedes Modul löst anfallende Probleme selbstständig ohne Auswirkungen auf das restliche Programm.

Unterschiedliche Modelle und andere Einstellungen des gleichen Wortartmarkierers werden über Vererbung implementiert. Die grundsätzlichen Methoden werden in einer abstrakten Oberklasse implementiert, sodass nur Details in erbenden Unterklassen festgelegt werden. Sie werden, da sie deshalb im gleichen Modul implementiert werden, gemeinsam geladen und ausgeführt. Der Stanford-Markierer und OpenNLP wurden so implementiert.

Das Implementieren neuer Wortartmarkierer bei denen eine Java-API zur Verfügung steht ist trivial. Im Standardkonstruktor wird der Wortartmarkierer über die API erstellt und in der `tag()`-Funktion genutzt, um ein Dokument zu markieren. Alle Wortartmarkierer bis auf Senna, TreeTagger und dem Goldstandardmarkierer wurden so implementiert.

Die Implementierung des TreeTaggers verwendet TreeTagger for Java[4] als Adapter.

Wortartmarkierer, die nur über Kommandozeilenaufrufe angesprochen werden, nutzen die `ProcessBuilder`-Architektur Der Wortartmarkierer wird als neuer Prozess gestartet und

[4]Siehe `https://reckart.github.io/tt4j/`, zuletzt abgerufen am 22.03.2016.

bekommt die nötigen Parameter als Argumente übergeben. Die Ausgabe des Prozesses wird umgeleitet und eine `TagContainer`-Liste erstellt, die dem `Document` übergeben wird. Senna wird wo implementiert.

Ein weiterer Spezialfall ist der Goldstandardmarkierer, der `Document` mit den korrekten Markierungen versehen soll. Es besteht die Möglichkeit die Markierungen bereits beim Einlesen des Goldstandards zu übernehmen. Da aber `tagman` nie die korrekten Markierungen benötigt, wurde das Setzen der korrekten Markierungen als Wortartmarkierer implementiert. Der Goldstandardmarkierer benötigt dazu den, Pfad der mit dem Goldstandard markierten Texten. Diese Information wird im `Document` vom Tokenisierer gespeichert. Der Goldstandartmarkierer liest aus dem Goldstandard die Markierungen und kopiert sie in `Document`. Falls die Datei keinen Goldstandard im korrekten Format enthält, wird eine Fehlermeldung ausgegeben.

Da die Wortarten als `String` gespeichert werden, sollten alle Wortartmarkierer die Wortartmarkierungen vereinheitlichen, um eine Vergleichbarkeit zu ermöglichen. Dies kann durch eine Übersetzung mit Hilfe eines Wörterbuchs vorm Schreiben stattfinden. Verwendet ein einzelner Wortartmarkierer einzigartige Markierungen, so werden diese Markiererintern übersetzt.

5.2.3.3. Dateiausgabemodule

Die konkreten `Writer`-Module sind für die Speicherung der `Document`-Datenstruktur zuständig. Um das Programm als Mehrfachwortartmarkierer verwenden zu können, speichert der `SimpleTxtWriter` die Markierungen eines jeden Wortartmarkierers separat, in dem in Abschnitt 4.2 definierten Format.

Die Markierungsergebnisse werden zur Weiterverarbeitung gemeinsam in einer `csv`-Datei gespeichert. Da die `csv`-Datei von unterschiedlichen Programmen genutzt wird, existieren zwei Versionen und damit zwei verschiedene `Writer`.

Um die Ergebnisse in einem Tabellenkalkulationsprogramm verwenden zu können, werden, mit Hilfe eines Wörterbuches, bestimmte Zeichenketten umgeschrieben und im `csv`-Format durch `SimpleCsvWriter` gespeichert. Als Separatorsymbol wird „;“ verwendet. Die Genauigkeit der Wortartmarkierer wird einfachheitshalber mit dem Tabellenkalkulationsprogramm berechnet. Ein weiterer Vorteil dieser Ausgabe ist die Möglichkeit über Tabellenkalkulationsprogramme die fehlerhaften Markierungen leicht deutlich hervorzuheben.

Für die Weiterverwendung der Ergebnisse im `claman`-Programm, müssen die Daten in einem, von Weka lesbaren Format gespeichert werden. Die nötigen Änderungen werden vor dem Speichern von `Csv4Weka` mit einem Wörterbuch durchgeführt. Um die Auswirkungen von Wörtern auf das Ergebnis analysieren zu können, wurde `Csv4WekaNoWords` im selben Modul implementiert. `Csv4WekaNoWords` hat die selbe Funktionsweise wie `Csv4Weka`, lässt aber beim Schreiben die Worte weg. Beide Klassen nutzen „ ,“ als Separatorsymbol.

Als letzer Dateischreiber wurde aus Bequemlichkeit `SimpleTexWriter` implementiert. Dieser Ausgabeschreiber erstellt eine `Tex`-Datei, die eine Tabelle enthält, die aus Namen der Wortartmarkierer und der Genauigkeiten besteht. Es wird angenommen, dass das erste Wortartmarkierermodul die Wortarten korrekt markiert. Alle anderen Markierungen werden mit diesem abgeglichen und die Anzahl gleicher Markierungen gezählt.

Alle Dateischreiber werden unabhängig von der Implementierung in `TaggingManager` in einer Warteschlange verwaltet und als letztes ausgeführt.

5.3. Claman-Unterprojekt

Das `Claman`-Unterprojekt ist dafür zuständig Klassifikationsverfahren auf dem Ausgabedatensatz von `tagman` zu trainieren, zu klassifizieren und zu evaluieren. Die trainierten

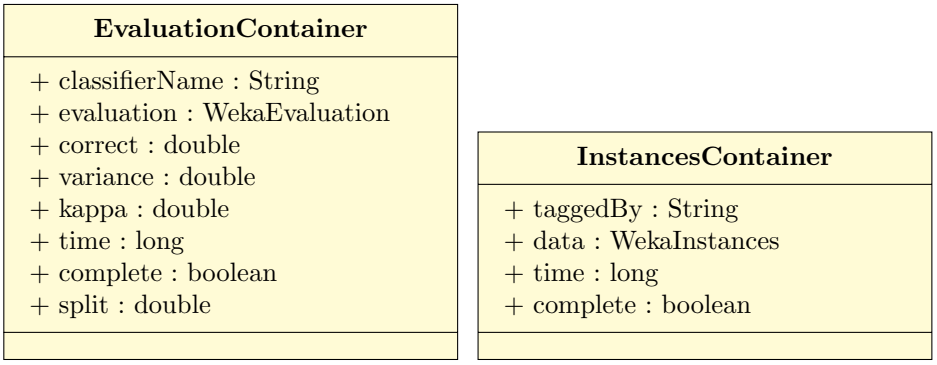

Abbildung 5.5.: EvaluationContainer und InstancesContainer

Modelle sollen gespeichert und später wieder geladen und verwendet werden können. Das Unterprojekt ist dabei stark von Weka abhängig.

Anders als bei `tagman` gibt es bei der Implementierung von `claman` einige Änderungen zum Entwurf.

5.3.1. Datenstruktur

Durch die Verwendung der Weka-API kann auf die dort verfügbaren Datenstrukturen und Methoden zurückgegriffen werden. Folgende Datenstrukturen werden aus Weka importiert.

Um den Datensatz darzustellen wird `Instances` verwendet. Auf dieser Datenstruktur arbeiten die Klassifikationsverfahren. Sie verwenden `Instances` zum trainieren, klassifizieren und evaluieren. `ClassifikationManager` liest die `csv`-Ausgabedatei von `tagman` und erzeugt daraus ein `Instances`-Objekt. In `Instances` kann ein Attribut als Klasse definiert werden.

Wird ein Klassifikationsverfahren durch Weka evaluiert, so werden die Ergebnisse in `Evaluation` gespeichert. Leider wird dabei nicht das Klassifikationsverfahren gespeichert, was nötig wäre, um das Ergebnis der Evaluation mit dem Klassifikationsverfahren ausgeben zu können. Deshalb wurde, anders als im Entwurf spezifiziert, `EvaluationContainer` hinzugefügt, um diese Information ebenfalls zu speichern. Da dieses Problem auch bei `Instances` besteht, wurde ebenfalls eine Klasse `InstancesContainer` hinzugefügt, die die gleiche Funktion erfüllt. Beide Containerklassen sind in Abbildung 5.5 dargestellt.

5.3.2. ClassificationManager-Hauptklasse

`ClassificationManager` ist die zentrale Steuerklasse des `claman`-Unterprojekts. Hier werden die Module durch die `ServiceLoader`-Architektur geladen, gesammelt und verwalten. Anders als bei `tagman` gibt es keine Einlesemodule. `ClassificationManager` liest selbständig die Datei über die Methode `createInstances(File file)` ein und wandelt sie mit Hilfe der Weka-API in `Instances` um. Die eingelesenen `Instances` werden über Filtermodule vorverarbeitet. Wie im Entwurf festgelegt, hängt der weitere Ablauf des Programmes nun vom Betriebsmodus ab.

Ist das Programm im „Modell trainieren“-Modus, so trainiert jedes Klassifikationsmodul auf dem aktuellen `Instances` ein Modell und speichert das Modell ab. Im Modus „Daten klassifizieren“ werden die zuvor trainierten Modelle geladen und darüber `Instances` klassifiziert. Der klassifizierte Datensatz wird nun ausgegeben.

Im letzten Modus werden die Klassifikationsverfahren anhand `Instances` evaluiert. Während es im Entwurf geplant war, dass nur eine k-fache Kreuzvalidierung durchgeführt

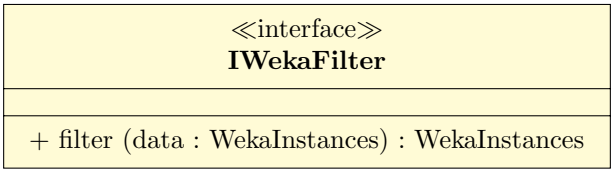

Abbildung 5.6.: *IWekaFilter*-Schnittstelle für Filter

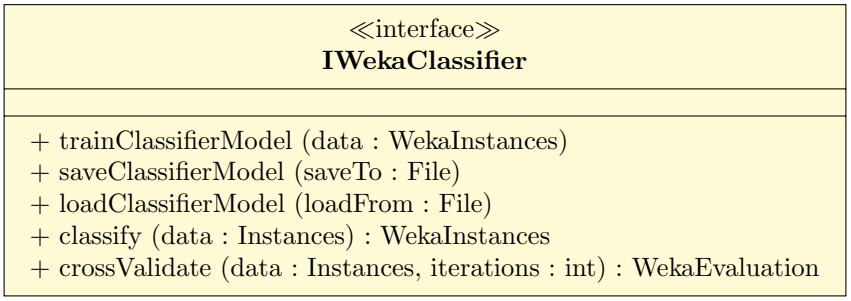

Abbildung 5.7.: *IWekaClassifier*-Schnittstelle zur Implementierung der Klassifikationsverfahren

wird, wurde auch die Monte-Carlo-Kreuzvalidierung implementiert. Sie kann als alternativer Evaluationsmodus aufgerufen werden. Die Ergebnisse der Evaluation werden über entsprechende Datenschreiber ausgegeben. Während der Evaluation werden Modelle mehrmals trainiert und getestet.

5.3.3. Modulschnittstellen

Über die *IWekaFilter*-Schnittstelle, die in Abbildung 5.6 dargestellt ist, werden Weka-Filter implementiert und gesammelt. Die Filter ermöglichen eine Vorverarbeitung des Datensatzes vor der eigentlichen Klassifikation.

Die in Abbildung 5.7 dargestellte *IWekaClassifier*-Schnittstelle wird von den entsprechenden Weka-Klassifikatoren implementiert. Sie bieten die Möglichkeit Modelle zu trainieren, zu speichern, zu laden und zum Klassifizieren zu nutzen. Die Ausgabe der Klassifizierung sind, wie auch bei der Eingabe, `Instances` der Weka-API. Weiterhin müssen sie ein Modell durch Kreuzvalidierung evaluieren können. Die nötigen Einstellungen der Iterationsanzahl und des Trainings- und Testdatentrennungsverhältnis werden ihnen übergeben. Der Rückgabewert der Kreuzvalidierung ist ein von der Weka-API bereitgestelltes `Evaluation`-Objekt, welches die verschiedenen Evaluierungsergebnisse gekapselt hat.

Die Schnittstelle ermöglicht es weiterhin Optionen für die Klassifikationsverfahren zu setzen.

Da die Weka-API genutzt wird, unterscheidet sich die konkrete Implementierung der Klassifikationsverfahren, nur im genutzten Klassifikator. Ein Großteil der Implementierung befindet sich daher in der abstrakten Klasse `AbstractClassifier`. Die Module erben von `AbstractClassifier` und setzen nur einen Kurznamen und den konkret genutzten Klassifikator.

Wurde, mit Hilfe eines vorher trainierten Modells, ein Datensatz klassifiziert, so findet eine Ausgabe der veränderten `Instances`-Daten über eine in Abbildung 5.8 dargestellte *IWekaInstaceWriter*-Schnittstelle statt. Die Weka-API bietet Methoden, um `Instance`-Objekte in das `csv`-Format schreiben zu können. Dies wird vom `SimpleCsvInstanceWriter` genutzt.

Abbildung 5.8.: `IWekaInstancesWriter`-Schnittstelle zur Ausgabe der klassifizierten Daten

Abbildung 5.9.: *`IWekaEvaluationWriter`*-Schnittstelle zur Ausgabe der Evaluationsergebnisse

Läuft das Programm im „Kreuzvalidierungs“-Betriebsmodus, so müssen die Ergebnisse ausgegeben werden. Die *`IWekaEvaluationWriter`*-Schnittstelle in Abbildung 5.9 erlaubt eine Ausgabe der Evaluation in ein entsprechendes Dateiformat. Bei der Ausgabe sollen die in Unterabschnitt 4.4.4 definierten Metriken tabellarisch, mit Namen des Klassifikators, gespeichert werden.

5.3.3.1. Filtermodule

Ursprünglich war es gedacht, dass die Wörter aus dem Datensatz durch einen Filter entfernt werden sollten. Dadurch sollte analysiert werden wie groß die Auswirkung der Wörter auf das Ergebnis der Kombination ist. Es hat sich aber als praktischer erwiesen, die Wörter bei der Ausgabe der Wortartmarkierer wegzulassen. Deshalb existiert zwar eine Schnittstelle für Filter, es wurden aber keine konkreten Filtermodule erstellt.

5.3.3.2. Klassifikationsmodule

Bei der Implementation der Klassifikationsverfahren gab es einige Änderungen zum Entwurf. Dadurch dass alle Weka-Implementierungen von der Weka-Klasse `Classifier` erben, können die Verfahren leicht implementiert werden. Alle nötigen Methoden für die drei Betriebsmodi wurden in `AbstractClassifier` der `claman-api` implementiert. Die konkreten Module implementieren daher nur noch die konkrete Weka-Implementierung des Klassifikationsverfahren. In Tabelle 5.1 sind die Weka-Implementierungen mit dem Klassifikationsverfahren aufgelistet. Für jedes Verfahren wurde ein Modul erstellt.

Eine Änderung zum Entwurf entstand durch die Containerklassen für `Instances` und `Evaluation`. Die *`IWekaClassifier`*-Schnittstelle nutzt nun `InstancesContainer`. Im Modus `Kreuzvalidierung` wird nun *`EvaluationContainer`* zurückgegeben.

Ursprünglich war es geplant nur die von Weka zur Verfügung gestellte k-fache Kreuzvalidierung zu nutzen. Um aber evaluieren zu können wie der Zusammenhang zwischen Größe des Trainingskorpus und Qualität des Ergebnisses ist, wurde auch das Monte Carlo Kreuzvalidierungsverfahren implementiert. Die *`IWekaClassifier`*-Schnittstelle wurde daher um die Methode `monteCarloCrossValidate(Instances data, int iterations, int percentage)` erweitert. Als Argumente dient hierbei, neben dem Datensatz, die Anzahl der Wiederholungen und die Anteilverteilung zwischen Trainings- und Testmenge. Die Ergebnisse der Evaluation werden direkt in `EvaluationContainer` geschrieben ohne eine `Evaluation` anzulegen.

Tabelle 5.1.: Implementierte Weka-Klassifikationsverfahren

Verfahren	Kurzbeschreibung
BayesNet	Bayessches Netz
HyperPipes	Für jede Klasse wird eine Regel abgeleitet
IBk	k-nächste-Nachbar-Methode (kNN)
J48	C4.5-Entscheidungsbaum
LibSVM	Stützvektormaschine (SVM)
NaiveBayes	Bayes-Klassifikator
OneR	1R-Algorithmus
PART	Regeln aus Entscheidungsbaum ableiten
REPTree	Einfacher Entscheidungsbaum
VFI	Merkmalsintervallsauswahl
NNge	kNN-ähnliches Verfahren mit Regeln
JRip	Regeln mit wiederholten inkrementierten Stutzen
KStar	Instanzbasierter Lerner
Ridor	Ausnahmebasierte Regeln
AODE	Durchschnitt alternierender Bayes-Klassifikatoren
ib1	1-Nächster-Nachbar-Verfahren
Logistic	Multinominale Logische Regression
MultilayeredPerceptron	Mehrschichtiges Perceptron
RBFNetwork	Logische Regression mit Clustering für Basisfunktion
SimpleCart	Entscheidungsbaum, stutzt nach minimaler Kostenkomplexität
SMO	SVM mit sequenzieller Mimimaloptimierung
WAODE	Gewichtetes AODE

Ein Spezialfall war die Implementierung von LibSVM. Das Modul braucht nicht nur Weka, sondern auch die LibSVM-Bibliothek[5], um Lauffähig zu sein. Die Bibliothek wurde der lokalen Ablage hinzugefügt und über Maven als Abhängigkeit festgelegt.

Unabhängig vom Modus verwaltet eine Warteschlange die vom `ServiceLoader` geladenen *`IWekaClassifier`*.

5.3.3.3. Dateiausgabemodule

Wie in Unterabschnitt 5.3.3 festgelegt, gibt es zwei generelle Ausgabeschreiber. Die Schreiber werden abhängig vom Modus aufgerufen. Findet eine Klassifizierung statt, so werden die markierten `Instences` ausgegeben. Im Evaluationsmodus werden hingegen die Ergebnisse der `Evaluation` gespeichert.

Die Schreiber werden in entsprechenden Warteschlangen verwaltet.

Instances:

Die Weka-API bietet Methoden zum Speichern von `Instances` an. Diese werden vom `SimpleInstanceWriter` verwendet, um die Daten im csv-Format auszugeben. Dies ist die einzige konkrete Implementierung, um `Instances` auszugeben.

Evaluation:

Um dem Benutzer die Ergebnisse der Evaluation zur Verfügung zu stellen wurden zwei Ausgabeschreiber für `Evaluation` implementiert. Der erste ist `TexEvaluationWriter`. Er

[5]Siehe `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`, zuletzt abgerufen am 28.03.2016

gibt das Ergebnis der Evaluation in einer Tex-Tabelle aus. Neben dem Erwartungswert, der Varianz und Cohens Kappa wird auch die Dauer der Evaluation und die Verteilung der Trainings- und Testmengen ausgegeben.

Der `SummaryWriter` nutzt die Weka-API, um die `Evaluation` zusammenfassend in eine Textdatei zu schreiben. Dadurch gibt er mehr Metriken als `TexEvaluationWriter` aus. Da das Monte-Carlo-Kreuzvalidierungsverfahren eine Eigenimplementation ist, musste hier für eine eigene `toSummaryString()`-Methode implementiert werden. Das Verfahren gibt daher die gleichen Metriken wie `TexEvaluationWriter` aus.

5.4. Zusammenfassung

In diesem Kapitel wurden die Probleme und Herausforderungen besprochen, die während der Implementierung des Entwurfs aus Abschnitt 4.4 aufkamen. Auch wurde der gesamte Aufbau des Projektes, die konkret implementierten Module und die genutzten Werkzeuge und Bibliotheken erläutert. Zwischen Entwurf und Implementierung gab es einige Erweiterungen und Änderungen. Die größte dieser Änderungen ist, dass die Evaluation bei `ClassificationManager` den Monte-Carlo-Kreuzvalidierungsmodus zusätzlich nutzen kann. Dies ermöglicht eine alternative Evaluation der Arbeit und die Analyse der Auswirkung der Größe des Trainingskorpus auf die Ergebnisse. Weiterhin war es nötig in `claman` Containerklassen für `Instances` und `Evaluation` zu verwenden. Um Ergebnisse leichter darzustellen wurden außerdem weitere Untermodule implementiert.

6. Evaluation

Das Ziel dieser Arbeit ist es Wortartmarkiererergebnisse durch die Nutzung von Klassifikationsverfahren zu optimieren. Die Schritte die nötig sind, um dieses Ziel zu erreichen, wurden in Kapitel 4 erläutert. Daraus wurde eine softwaretechnische Lösung entworfen und implementiert. Der Entwurf aus Abschnitt 4.4 sieht zwei separate Programme vor, die beide jeweils einen Teilschritt zur Lösung des Problems beitragen. Das erste Programm koordiniert Wortartmarkierer aus Abschnitt 4.3, um Texte zu markieren. Die Ausgabe des ersten Programmes wird vom zweiten Programm genutzt, um mit Hilfe von Klassifikationsverfahren Modelle zu trainieren, Texte zu markieren oder die Verfahren zu evaluieren. Die genutzten Verfahren sind in Tabelle 6.1 kurz erläutert. Die Tabelle ist identisch mit Tabelle 5.1 aus Kapitel 5, aber die Verfahren fehlen, die aufgrund zu hohem Arbeitsspeicherbedarfs oder zu langer Rechenzeit nicht evaluiert werden konnten.

Evaluiert wird in diesem Kapitel auf den drei englischsprachigen Korpora aus Abschnitt 4.2. Die Korpora und einige ihrer Eigenschaften sind in Tabelle 6.2 aufgelistet.

Als Metriken für die Evaluation dienen der Erwartungswert der Genauigkeit, die Varianz des Erwartungswertes und Cohens Kappa. Diese Metriken wurden in Abschnitt 2.4 erläutert.

In Unterabschnitt 4.3.6 wurde eine Performanzanalyse der verwendeten Wortartmarkierer durchgeführt. Auf jedem Korpus konnte so der beste Wortartmarkierer gefunden werden. Die Verfahren werden mit diesem Wortartmarkierer verglichen, um Festzustellen, ob eine Verbesserung durch das Verfahren stattfand. In Tabelle 6.2 stehen die besten Wortartmarkierer mit ihrer Genauigkeit als Vergleichsmarkierer dar. Weiterhin sollen die Verfahren auch mit der einfachen Mehrheitswahl verglichen werden. Dabei besitzt jeder Wortartmarkierer eine Stimme und dem Wort wird die Markierung zugewiesen, die von den meisten Markierern Zuspruch bekommt.

Zu Beginn des Kapitels werden Hypothesen aufgestellt, anhand deren die Ergebnisse evaluiert werden. Der Rest des Kapitels teilt sich durch die Verwendung der 10-fachen Kreuzvalidierung und Monte-Carlo-Kreuzvalidierung in zwei entsprechende Abschnitte auf. Jeder Abschnitt behandelt die Korpora einzeln und evaluiert die Ergebnisse des Verfahrens anhand der angenommen Hypothesen. Durch die 10-fache Kreuzvalidierung soll festgestellt werden, ob das gesamte Verfahren eine Verbesserung bringt. Mit Hilfe der Monte-Carlo-Kreuzvalidierung wird die Auswirkung der Größe des Trainingskoprpus auf die Genauigkeit untersucht.

Tabelle 6.1.: Evaluierte Klassifikationsverfahren mit Kurzbeschreibung

Verfahren	Kurzbeschreibung
BayesNet	Bayessches Netz
HyperPipes	Für jede Klasse wird eine Regel abgeleitet
IBk	k-nächste-Nachbar-Methode (kNN)
J48	C4.5-Entscheidungsbaum
LibSVM	Stützvektormaschine (SVM)
NaiveBayes	Bayes-Klassifikator
OneR	1R-Algorithmus
PART	Regeln aus Entscheidungsbaum ableiten
REPTree	Einfacher Entscheidungsbaum
VFI	Merkmalsintervallsauswahl
NNge	kNN-ähnliches Verfahren mit Regeln
JRip	Regeln mit wiederholten inkrementierten Stutzen
KStar	Instanzbasierter Lerner
Ridor	Ausnahmebasierte Regeln

Tabelle 6.2.: Auflistung der verwendeten Korpora

Korpuseigenschaften				Vergleichsmarkierer	
Korpus	Bereich	Worte	Stil	Name (Modell)	Genauigkeit
WSJ	Wirtschaft	94.085	Fachzeitschrift	Jitar	98,54%
NLCI	Erzählung	18.124	Skript	Stanford (bidir)	96,17%
PARSE	Alltag	2.174	Aufgabenerteilung	Stanford (bidir)	90,66%

Weiterhin soll auch die Auswirkung der Worte auf das Ergebnis analysiert werden. Da Worte Informationen enthalten, dürfte das Weglassen zu einem Verlust der Genauigkeit führen. Zeitgleich dürfte aber das Modell für die praktische Verwendung besser geeignet sein, da die Worte überbewertet werden könnten.

Die Rechenzeit der gesamten Evaluation betrug ungefähr 8,5 Tage.

6.1. Hypothesen

Aus der Analyse in Kapitel 4 lassen sich einige Hypothesen, anhand derer das gesamte Verfahren evaluiert wird, ableiten.

H_1 Die zentrale Hypothese ist, dass mindestens ein Verfahren besser abschneidet als der Vergleichsmarkierer.

H_2 Es wird mindestens ein Verfahren geben, das besser als die einfache Mehrheitswahl ist.

H_3 Jedes Klassifikationsverfahren führt zu einer Verbesserung.

H_4 Komplexere Verfahren schneiden besser ab als einfache Verfahren, da sie die zur Verfügung stehenden Informationen besser nutzen. Hierfür wird angenommen, dass komplexere Klassifikationsverfahren länger zur Evaluation benötigen als einfache Verfahren.

H_5 Die Differenz der Genauigkeiten δ zwischen den Korpora verringern sich.

H_6 An der Anordnung der δ aus Hypothese H_6 zwischen den Korpora gibt es keine Änderung.

Tabelle 6.3.: Evaluation des WSJ-Korpus mit Worten, sortiert nach Erwartungswert.

Name	Komplexität	Erwartungswert	Varianz	Kappa	Dauer
J48	Halbkomplex	98,85%	0,0183	0,9876	34,59s
PART	Halbkomplex	98,70%	0,0187	0,986	546,92s
REPTree	Halbkomplex	98,69%	0,0196	0,9859	130,70s
JRip	Komplex	98,62%	0,0201	0,9851	4139,03s
Jitar		98,54%	Vergleichsmarkierer		
IBk	Halbkomplex	98,54%	0,0200	0,9843	814,03s
OneR	Einfach	98,53%	0,0216	0,9842	0,72s
KStar	Komplex	98,50%	0,0201	0,9839	14636,95s
Einfache Mehrheitswahl		98,43%	Vergleichsverfahren		
Ridor	Komplex	98,36%	0,0228	0,9824	27795,12s
BayesNet	Einfach	98,25%	0,0231	0,9812	5,40s
NaiveBayes	Einfach	98,20%	0,0234	0,9807	6,01s
NNge	Komplex	98,05%	0,0247	0,9791	7930,22s
LibSVM	Komplex	95,91%	0,036	0,956	5765,17s
VFI	Einfach	95,80%	0,0409	0,955	3,00s
HyperPipes	Einfach	71,65%	0,1115	0,6951	0,86s

H_7 Werden Worte als Attribute den Klassifikationsverfahren zur Verfügung gestellt, so führt das zu besseren Ergebnissen.

H_8 Wird nur auf den Markierungen der Wortartmarkierer ohne Worte klassifiziert, so wird es trotzdem mindestens ein Verfahren geben, das besser ist als der Vergleichsmarkierer und die einfache Mehrheitswahl.

H_9 Wie auch bei den Wortartmarkierern führen mehr Trainingsdaten zu einem besseren Ergebnis.

6.2. 10-fache Kreuzvalidierung

Bei der 10-fachen Kreuzvalidierung wird der gesamte Korpus in zehn Teilmengen gespalten. Reihenweise wird eine Teilmenge zum Testen ausgewählt und auf den restlichen neun wird das Modell trainiert. Dies wird wiederholt bis auf jeder Teilmenge getestet wurde.

Anhand der Ergebnismetriken der Kreuzvalidierung kann festgestellt werden, ob eine Verbesserung stattfand. Gibt es Verfahren, die einen höheren Erwartungswert der Genauigkeit haben als der Vergleichsmarkierer, so ist das gesamte Verfahren erfolgreich. In den Tabellen der Korpora sind die Verfahren nach dem Erwartungswert der Genauigkeit sortiert. Neben den Klassifikationsverfahren ist auch jeweils der Vergleichsmarkierer und die einfache Mehrheitswahl eingetragen. Alle Verfahren über dem Vergleichsmarkierer optimieren daher erfolgreich. Ist die einfache Mehrheitswahl bereits besser als der Vergleichsmarkierer, so wird diese als Vergleich genutzt. Die einfache Mehrheitswahl ist die einfachste Möglichkeit Wortartmarkierer zu kombinieren. Dadurch sind Verfahren ungenügend, die schlechter als die einfache Mehrheitswahl abschneiden, selbst wenn sie besser als der Vergleichsmarkierer sind.

Findet eine Verbesserung durch das Klassifikationsverfahren statt, so soll auch festgestellt werden wie groß diese ist.

6.2.1. WSJ-Korpus

Die Ergebnisse der 10-fachen Kreuzvalidierung auf dem WSJ-Korpus sind in Tabelle 6.3 dargestellt. Die einfache Mehrheitswahl schneidet dabei schlechter ab, als der Vergleichs-

markierer. Eine Ursache hierfür ist, dass der Vergleichsmarkierer an diesen Korpus überangepasst ist. Da dies vermutlich bei den anderen Markierern nicht zutrifft, hat der überangepasste Vergleichsmarkierer genausoviel Einfluss wie die schlechten Markierer. Die einfache Mehrheitswahl ist aber deutlich besser, als die restlichen Wortartmarkierer. Aus der Tabelle ist ersichtlich, dass es mehrere Verfahren gibt, die den Vergleichsmarkierer übertreffen. Die besten drei Plätze werden von Verfahren belegt, die auf Entscheidungsbäumen basieren. J48 konnte als bestes Verfahren den Vergleichsmarkierer Jitar um 0,41 Prozentpunkte übertreffen. Dies ist besonders interessant, da Jitar auf dem WSJ-Korpus trainiert wurde. Das heißt, dass es Verfahren gibt, die selbst einen möglicherweise überangepassten Wortartmarkierer verbessern. Die meisten Verfahren schneiden aber schlechter ab als der Vergleichsmarkierer. Bis auf drei Ausreißer, haben aber auch die Klassifikationsverfahren eine Genauigkeit von über 98,00%. Eine Erklärung dafür, dass die Verfahren das Ergebnis verschlechtern könnte sein, dass Wert auf stark spezifische Attribute gelegt wird. Wird beispielsweise zu viel Wert auf Worte gelegt, so nähert sich das Verfahren einem Wortartmarkierer an, der den Worten die meist verwendete Wortart zuweist. Das Ergebnis von 92,48% von HyperPipes ist sehr schlecht. Bereits einfache Wortartmarkierer überschreiten diesen Wert stark. Das schlechte Abschneiden von HyperPipes ist auch weiterhin interessant, da HyperPipes als eine Erweiterung des OneR-Algorithmus gilt und es erwartet wurde, dass HyperPipes zumindest OneR übertrifft. Da OneR schlechter als Jitar abschneidet, ist anzunehmen, dass die Ergebnisse des besten Vergleichsmarkierer nicht einfach übernommen werden.

Da die Verfahren unterschiedlich komplex sind, sollen sie in drei Komplexitätsklassen eingeteilt werden. Dadurch ist es möglich Aussagen über ähnliche Verfahren zu treffen. Einfache Verfahren konnten in weniger als 30 Sekunden evaluiert werden. Komplexe Verfahren brauchen hingegen für die gesamte Evaluation länger als fünfzehn Minuten. Alle weiteren Verfahren werden in die Klasse Halbkomplex eingeordnet. Mit Hilfe dieser Anordnung lassen sich die durchschnittlichen Ergebnisse der Komplexitätsklassen berechnen. Aus Tabelle 6.4 ist ersichtlich, dass die halbkomplexen Verfahren nicht nur mit J48 das beste Verfahren beinhalten, sondern auch mit 98,70% durchschnittlich am besten sind. Weiterhin sind die drei besten Verfahren in diese Kategorie einzuordnen. Die einfachen Verfahren versagen, mit ihrem besten Verfahren OneR, sogar knapp mit 98,53% gegen den Vergleichsmarkierer. Am schlechtesten von allen Klassen schneiden sie mit 92,49% ab. Das beste komplexe Verfahren JRip ist mit 98,62% besser als der Vergleichsmarkierer, aber auch diese Verfahren können die halbkomplexen Verfahren mit 97,89% durchschnittlich nicht übertreffen. Zwar sind einfache Verfahren am schlechtesten, aber die halbkomplexen Verfahren sind den komplexen Verfahren überlegen. Ein Einwand zu diesem Ergebnis könnte sein, dass einige Verfahren aufgrund nicht ausreichendem Arbeitsspeicher oder zu langer Evaluationsdauer nicht im Ergebnis einberechnet werden konnten. Es wäre aber auch dann schwer eine Verbesserung von 0,75 Prozentpunkten der halbkomplexen Verfahren, gegenüber den komplexen Verfahren, zu überholen.

Um die Auswirkungen der Worte als Attribute auf das Gesamtergebnis einschätzen zu können, wurde das Attribut „Wort“ entfernt. In Tabelle 6.5 sind die Ergebnisse der 10-fachen Kreuzvalidierung ohne Worte dargestellt. Da Wörter auf die einfache Mehrheitswahl keinen Einfluss haben, ist das Vergleichsverfahren hier ebenfalls schlechter als der Vergleichsmarkierer. Das Klassifikationsverfahren J48 ist, wie auch auf dem Korpus mit Worten, mit einer Genauigkeit von 98,73% das beste Verfahren. Es übertrifft damit den Vergleichsmarkierer um 0,30 Prozentpunkte, schneidet aber gegenüber dem Korpus mit Worten, um -0,12 Prozentpunkte schlechter ab. Die vier Verfahren, die bessere Ergebnisse erzielen als der Vergleichsmarkierer, bleiben weiterhin gleich. Wie in Tabelle 6.6 ersichtlich, hat das Fehlen der Wörter bei den meisten Verfahren kaum Auswirkungen auf das Ergebnis. Einzig die beiden instanzbasierten Verfahren LibSVM (2,49 Prozentpunkte Verbesserung) und NNge

Tabelle 6.4.: Komplexitätsklassen mit der Genauigkeit des besten Verfahrens und der durchschnittlichen Genauigkeit auf den Korpora.

	Einfach		Halbkomplex		Komplex	
Korpus	bestes	durch.	bestes	durch.	bestes	durch.
WSJ - Worte	98,53%	92,49%	98,85%	98,70%	98,62%	97,89%
WSJ - ohne W.	98,52%	83,12%	98,73%	98,65%	98,61%	98,44%
NLCI - Worte	98,59%	97,26%	99,11%	98,48%	99,05%	98,39%
NLIC - ohne W.	98,03%	95,86%	98,95%	98,81%	98,60%	98,23%
PARSE - Worte	97,88%	96,60%	98,57%	97,94%	98,34%	96,81%
PARSE - ohne W.	95,62%	90,93%	97,62%	98,07%	98,66%	98,08%

Tabelle 6.5.: Evaluation auf WSJ-Korpus ohne Worte, sortiert nach Erwartungswert.

Name	Einfach	Erwartungswert	Varianz	Kappa	Dauer
J48	Halbkomplex	98,73%	0,0188	0,9864	4,74s
REPTree	Halbkomplex	98,70%	0,0186	0,9861	9,14s
PART	Halbkomplex	98,64%	0,019	0,9854	16,55s
JRip	Komplex	98,61%	0,0202	0,985	453,97s
Jitar		98,54%	Vergleichsmarkierer		
IBk	Halbkomplex	98,53%	0,0201	0,9843	817,30s
OneR	Einfach	98,52%	0,0216	0,9842	0,68s
KStar	Komplex	98,47%	0,0203	0,9836	12906,79s
NNge	Komplex	98,44%	0,0222	0,9832	351,96s
Einfache Mehrheitswahl		98,43%	Vergleichsverfahren		
LibSVM	Komplex	98,40%	0,0225	0,9828	1059,22s
Ridor	Komplex	98,28%	0,0234	0,9815	901,49s
NaiveBayes	Einfach	98,25%	0,0231	0,9813	5,94s
BayesNet	Einfach	98,22%	0,0232	0,9809	3,97s
VFI	Einfach	88,27%	0,0623	0,8755	2,58s
HyperPipes	Einfach	32,33%	0,1126	0,2664	0,74s

Tabelle 6.6.: Auswirkung von Worten auf das Ergebnis auf dem WSJ-Korpus

Verfahren	Komplexität	Mit Worten	Ohne Worte	δ in %Punkten
J48	Halbkomplex	98,85%	98,73%	-0,12
PART	Halbkomplex	98,70%	98,64%	-0,06
REPTree	Halbkomplex	98,69%	98,70%	0,01
JRip	Komplex	98,62%	98,61%	-0,01
IBk	Halbkomplex	98,54%	98,53%	-0,01
OneR	Einfach	98,53%	98,52%	-0,01
KStar	Komplex	98,50%	98,47%	-0,03
Ridor	Komplex	98,36%	98,28%	-0,08
BayesNet	Einfach	98,25%	98,22%	-0,03
NaiveBayes	Einfach	98,20%	98,25%	0,05
NNge	Komplex	98,05%	98,44%	0,39
LibSVM	Komplex	95,91%	98,40%	2,49
VFI	Einfach	95,80%	88,27%	-7,53
HyperPipes	Einfach	71,65%	32,33%	-39,32

Tabelle 6.7.: Evaluation des NLCI-Korpus mit Worten, sortiert nach Erwartungswert.

Name	Komplexität	Erwartungswert	Varianz	Kappa	Dauer
IBk	Halbkomplex	99,11%	0,0200	0,9900	10,99s
KStar	Komplex	99,05%	0,0207	0,9893	511,07s
NNge	Komplex	98,97%	0,0234	0,9884	45,80s
PART	Halbkomplex	98,74%	0,0244	0,9858	1,73s
J48	Halbkomplex	98,69%	0,0252	0,9852	0,50s
JRip	Komplex	98,64%	0,0253	0,9848	29,79s
VFI	Einfach	98,59%	0,0302	0,9841	0,32s
BayesNet	Einfach	98,46%	0,0283	0,9828	0,49s
Ridor	Komplex	98,40%	0,0294	0,982	50,86s
NaiveBayes	Einfach	98,12%	0,031	0,979	0,67s
Einfache Mehrheitswahl		97,65%	Vergleichsverfahren		
REPTree	Halbkomplex	97,39%	0,0301	0,9707	1,90s
LibSVM	Komplex	96,89%	0,041	0,965	129,35s
OneR	Einfach	96,35%	0,0443	0,959	0,07s
Stanford (bidir)		96,17%	Vergleichsmarkierer		
HyperPipes	Einfach	94,77%	0,1200	0,9407	0,08s

(0,39 Prozentpunkte Verbesserung) können ohne Worte ein signifikant höheres Ergebnis erreichen. Es kann vermutet werden, dass bei instanzbasierten Verfahren die Worte zu viel Auswirkungen auf das Ergebnis haben. Unbekannte und mehrdeutige Worte werden daher öfters falsch klassifiziert als bei einer reinen Klassifizierung nach Markierungen. Beide Verfahren konnten aber den Vergleichsmarkierer nicht übertreffen.

Bei den beiden Verfahren Hyperpipes (-39,32 Prozentpunkte) und VFI (-7,53 Prozentpunkte) scheinen sehr stark von Worten abzuhängen, was zu einer starken Verschlechterung der Genauigkeiten führt. Bei Hyperpipes kann angenommen werden, dass Regeln gebildet wurden, bei denen knapp die Hälfte aller Klassen nur von den Worten abhängt. Dies führt dazu, dass HyperPipes das schlechte Ergebnis mit Worten stark unterbietet und mit einer Genauigkeit von 32,33% das schlechteste Verfahren überhaupt ist.

Beim Thema Komplexität fällt deutlich auf, dass die komplexen Verfahren vom Weglassen der Worte am meisten profitieren. Während sie mit Worten bei einer durchschnittlichen Genauigkeit von 97,89% sind, erhöht sich die Genauigkeit ohne Worte um 0,55 Prozentpunkte auf 98,44%. Eine Erklärung hierfür wäre, dass durch die fehlenden Wörter die Verfahren auf andere Attribute Wert legen und es dadurch vermeiden rein aufgrund der Worte zu klassifizieren. Der starke Einfluss der Worte kann dazu führen, dass mehrdeutige Worte nicht erkannt und somit falsch klassifiziert werden. Bei den einfachen Verfahren hat das schlechte Abschneiden von HyperPiepes und VFI starke Auswirkungen auf das Durchschnittsergebnis der einfachen Verfahren. Während die Ergebnisse der anderen einfachen Verfahren nur geringe Veränderung aufweisen, führen HyperPipes und VFI dazu, dass der Durchschnitt der einfachen Verfahren stark sinkt. Mit durchschnittlich 83,12% sind sie die schlechtesten Verfahren. Die halbkomplexen Verfahren sind mit 98,65% weiterhin die besten Verfahren, obwohl geringe Verluste stattfanden. Die Distanz zwischen den halbkomplexen Verfahren und den komplexen Verfahren hat sich stark verringert.

6.2.2. NLCI-Korpus

Auch auf den NLCI-Korpus ist eine Verbesserung in Tabelle 6.7 festzustellen. Alle getesteten Verfahren, bis auf HyperPipes, führten zu einer Verbesserung. Da die einfache Mehrheitswahl ebenfalls den Vergleichsmarkierer übertreffen kann, gibt es einige Verfahren die

Tabelle 6.8.: δ der Ergebnisse zwischen den besten Markierern in Prozentpunkten

Markierer	WSJ	NLCI
NLCI	2,37	
PARSE	7,88	5,51

Tabelle 6.9.: δ der Ergebnisse zwischen den besten Verfahren in Prozentpunkten

Verfahren	WSJ	NLCI
NLCI	-0,26	
PARSE	0,28	0,54

ungenügend abschneiden. Anders als auf dem WSJ-Korpus erreicht HyperPipes hier ein den anderen Verfahren ähnliches Ergebnis. Drei Verfahren konnten sogar das beste Ergebnis auf dem WSJ-Korpus übertreffen. Das beste der Verfahren IBk ist die Implementierung eines kNN-Verfahrens und erreicht ein Ergebnis von 99,11%. Damit übertrifft dieses Verfahren den Vergleichsmarkierer um 2,94 Prozentpunkte. Anders als beim WSJ-Korpus sind die besten drei Verfahren instanzbasiert und die Entscheidungsbäume folgen darauf.

In Tabelle 6.8 ist die Differenz der Genauigkeit δ zwischen den Vergleichsmarkierern ersichtlich. Der Vergleichsmarkierer Jitar ist auf dem WSJ-Korpus um 2,37 Prozentpunkte besser als der Vergleichsmarkierer auf dem NLCI-Korpus. Das δ zwischen J48 (WSJ-Korpus) und IBk (NLIC-Korpus), dargestellt in Tabelle 6.9, beträgt hingegen -0,26 Prozentpunkte Damit fand nicht nur eine Verringerung des δ statt, sondern auch eine Umdrehung der Anordnung. Eine Ursache hierfür könnte eine Ähnlichkeit der Sätze in diesem Korpus und die geringe Korpusgröße sein. Weiterhin fand eine Verringerung des δ zwischen NLCI und PARSE-Korpus statt. Das δ zwischen den beiden Korpora fiel von 5,51 Prozentpunkte auf 0,54 Prozentpunkte.

Für die Komplexitätsevaluation wird die Einteilung der Verfahren in drei Klassen aus dem WSJ-Korpus übernommen. Das Ergebnis in Tabelle 6.4 ähnelt dem des WSJ-Korpus. Die halbkomplexen Verfahren sind mit 98,48% wieder am besten, während die einfachen Verfahren mit 97,26% am schlechtesten abschneiden. Beide Komplexitätsklassen schneiden schlechter ab als auf dem WSJ-Korpus Dies ist interessant, da Verfahren existieren, die die besten Verfahren des WSJ-Korpus übertreffen. Die komplexen Verfahren sind mit 98,39% knapp hinter den halbkomplexen Verfahren und schneiden auf diesem Korpus besser ab als auf dem WSJ-Korpus mit Worten und liegen nur knapp hinter dem WSJ-Korpus ohne Worte. Dies liegt vor allem am Verfahren LibSVM. Auf dem NLCI-Korpus ist eine deutlich stärkere Varianz der Genauigkeiten zwischen den Verfahren als auf dem WSJ-Korpus ersichtlich.

Die Ergebnisse der Evaluation des NLCI-Korpus ohne Worte sind in Tabelle 6.10 dargestellt. Auch hier ist das Verfahren IBk mit 98,95% das beste Verfahren. Durch das Weglassen der Wörter ist das Ergebnis um 0,16 Prozentpunkte schlechter. Die Verfahren LibSVM mit 97,36% und OneR mit 96,24% sind zwar beide besser als der Vergleichsmarkierer mit 96,17%, schneiden aber schlechter ab als der einfache Mehrheitsentscheid mit 97,65%.

In Tabelle 6.11 ist die Auswirkung der Worte auf das Ergebnis auf dem NLCI-Korpus ersichtlich. REPTree kann ohne Worte die höchste Verbesserung von 1,47 Prozentpunkten erreichen, gegenüber der Verwendung von Worten als Attribut. Alle anderen Verfahren, bis auf J48 (0,11 Prozentpunkte Verbesserung) zeigen negative Auswirkungen auf die Genauigkeit. Das J48-ähnliche auf Entscheidungsbäumen basierende Verfahren REPTree zeigt deutliche Verbesserung, weshalb es anzunehmen ist, dass die Verbesserung von J48, nicht der Varianz geschuldet ist, obwohl nur eine geringe Verbesserung stattfand. Ursache da-

Tabelle 6.10.: Evaluation des NLCI-Korpus ohne Worte, sortiert nach Erwartungswert.

Name	Komplexität	Erwartungswert	Varianz	Kappa	Dauer
IBk	Halbkomplex	98,95%	0,0227	0,9882	16,56s
REPTree	Halbkomplex	98,86%	0,0237	0,9872	0,72s
J48	Halbkomplex	98,80%	0,0244	0,9865	0,37s
PART	Halbkomplex	98,63%	0,0254	0,9847	1,46s
JRip	Komplex	98,60%	0,0265	0,9844	22,04s
KStar	Komplex	98,58%	0,0246	0,9841	465,94s
NNge	Komplex	98,46%	0,0284	0,9827	25,33s
Ridor	Komplex	98,17%	0,0314	0,9795	33,80s
BayesNet	Einfach	98,03%	0,0313	0,9778	0,44s
NaiveBayes	Einfach	97,98%	0,031	0,9772	0,68s
VFI	Einfach	97,69%	0,0558	0,9741	0,34s
Einfache Mehrheitswahl		97,65%	Vergleichsverfahren		
LibSVM	Komplex	97,36%	0,0377	0,9703	72,79s
OneR	Einfach	96,24%	0,0451	0,9577	0,08s
Stanford (bidir)		96,17%	Vergleichsmarkierer		
HyperPipes	Einfach	89,35%	0,1211	0,8788	0,08s

für scheint, dass die Entscheidungsbäume, bei Verwendung von Worten, diese als erste Verzweigung nutzen. Dadurch werden mehrdeutige Worte nur aufgrund des Wortes klassifiziert, wenn im Trainingskorpus das Wort nur ein einziges mal vorkommt. Werden die Worte entfernt, so wird anhand der Markierungen entschieden. Dies führt zu einem tieferen Baum, da mehrere Attribute genutzt werden müssen. HyperPipes (-5,42 Prozentpunkte) und VFI (-0,90 Prozentpunkte) zeigen auch hier den größten Verlust.

Wird die Komplexität in Tabelle 6.4 für den NLCI-Korpus ohne Worte betrachtet, so ist ersichtlich, dass die halbkomplexen Verfahren mit 98,81% nicht nur am besten abschneiden, sondern sogar die durchschnittliche Genauigkeit auf Worten um 0,33 Prozentpunkte übertreffen. Dies liegt vor allem am Verfahren REPTree, das eine Verbesserung von 1,47 Prozentpunkten, gegenüber dem Verfahren mit Worten, erreicht. Die restlichen halbkomplexen Verfahren zeigen geringe Veränderungen. Wie auch auf dem WSJ-Korpus haben die einfachen Verfahren mit einer durchschnittlichen Genauigkeit von 95,86% durch HyperPipes den größten Verlust. Die komplexen Verfahren schneiden mit 98,23%, anders als auf WSJ-Korpus, mit einer leichten Verschlechterung von -0,16 Prozentpunkten ab.

6.2.3. PARSE-Korpus

In Tabelle 6.12 ist das Evaluationsergebnis auf dem PARSE-Korpus dargestellt. Die einfache Mehrheitswahl kann auf dem PARSE-Korpus den Vergleichsmarkierer und LibSVM übertreffen. Alle untersuchten Klassifikationsverfahren führen zu einer Verbesserung der Wortartmarkierung. Das beste Verfahren ist, wie auch beim NLCI-Korpus, das kNN-Verfahren IBk mit 98,57% und einer Verbesserung von 7,91 Prozentpunkten im Vergleich zum Vergleichsmarkierer. Als zweitbestes Verfahren konnte sich PART mit 98,48% als entscheidungsbaumbasiertes Verfahren gegenüber KStar, einem weiteren instanzbasierten Verfahren, mit 98,34% durchsetzen. Keins der getesteten Verfahren schnitt schlechter als der Vergleichsmarkierer ab . Ein Grund hierfür könnte die geringe Größe und starke Ähnlichkeit der Sätze in diesem Korpus sein. Es kann festgestellt werden, dass die Differenz der Genauigkeiten δ zum WSJ-Korpus, wie auch zum NLCI-Korpus sank. Während das δ zwischen den Vergleichsmarkierern bei 7,88 Prozentpunkten lag, war es zwischen den Verfahren nur bei 0,28 Prozentpunkten. Der Abstand zum NLCI-Korpus ist damit größer als zum WSJ-Korpus.

Tabelle 6.11.: Auswirkung von Worten auf das Ergebnis auf dem NLCI-Korpus

Name	Komplexität	Mit	Ohne	δ in %Punkten
BayesNet	Einfach	98,46%	98,03%	-0,43
HyperPipes	Einfach	94,77%	89,35%	-5,42
IBk	Halbkomplex	99,11%	98,95%	-0,16
J48	Halbkomplex	98,69%	98,80%	0,11
JRip	Komplex	98,64%	98,60%	-0,04
KStar	Komplex	99,05%	98,58%	-0,47
LibSVM	Komplex	96,89%	97,36%	0,47
NNge	Komplex	98,97%	98,46%	-0,51
NaiveBayes	Einfach	98,12%	97,98%	-0,14
OneR	Einfach	96,35%	96,24%	-0,11
PART	Halbkomplex	98,74%	98,63%	-0,11
REPTree	Halbkomplex	97,39%	98,86%	1,47
Ridor	Komplex	98,40%	98,17%	-0,23
VFI	Einfach	98,59%	97,69%	-0,90

Tabelle 6.12.: Evaluation des PARSE-Korpus mit Worten, sortiert nach Erwartungswert.

Name	Komplexität	Erwartungswert	Varianz	Kappa	Dauer
IBk	Halbkomplex	98,57%	0,0326	0,9838	0,13s
PART	Halbkomplex	98,48%	0,0311	0,9827	0,19s
KStar	Komplex	98,34%	0,0368	0,9811	7,14s
NNge	Komplex	98,16%	0,0402	0,9791	1,54s
J48	Halbkomplex	97,97%	0,0400	0,9769	0,03s
JRip	Komplex	97,88%	0,0424	0,9758	0,95s
BayesNet	Einfach	97,88%	0,0393	0,9758	0,04s
VFI	Einfach	97,79%	0,0502	0,9751	0,03s
NaiveBayes	Einfach	97,37%	0,0419	0,9700	0,05s
Ridor	Komplex	96,73%	0,0534	0,9627	1,18s
REPTree	Halbkomplex	96,73%	0,0432	0,9625	0,07s
OneR	Einfach	95,48%	0,0635	0,9487	0,00s
HyperPipes	Einfach	94,47%	0,1397	0,9374	0,01s
Einfache Mehrheitswahl		93,01%	Vergleichsverfahren		
LibSVM	Komplex	92,95%	0,0791	0,9188	3,72s
Stanford (bidir)		90,66%	Vergleichsmarkierer		

Tabelle 6.13.: Evaluation des PARSE-Korpus ohne Worte, sortiert nach Erwartungswert.

Name	Komplexität	Erwartungswert	Varianz	Kappa	Dauer
NNge	Komplex	98,66%	0,0327	0,9847	0,97s
J48	Halbkomplex	98,62%	0,0344	0,9843	0,03s
KStar	Komplex	98,48%	0,0362	0,9827	6,69s
IBk	Halbkomplex	98,06%	0,0392	0,978	0,23s
JRip	Komplex	98,02%	0,0405	0,9773	0,73s
REPTree	Halbkomplex	97,88%	0,0413	0,9760	0,04s
PART	Halbkomplex	97,70%	0,0421	0,9739	0,09s
Ridor	Komplex	96,96%	0,0517	0,9656	1,15s
BayesNet	Einfach	95,62%	0,0589	0,9503	0,04s
NaiveBayes	Einfach	95,62%	0,0573	0,9502	0,05s
LibSVM	Komplex	95,53%	0,063	0,9489	2,73s
OneR	Einfach	93,04%	0,0789	0,9211	0,01s
Einfache Mehrheitswahl		93,01%	Vergleichsverfahren		
VFI	Einfach	91,98%	0,0835	0,9089	0,03s
Stanford (bidir)		90,66%	Vergleichsmarkierer		
HyperPipes	Einfach	78,39%	0,1404	0,7618	0,01s

Um die Auswirkung der Komplexität der Verfahren untersuchen zu können, werden die Komplexitätsklasseneinteilung des WSJ-Korpus übernommen. In Tabelle 6.4 sind wieder die besten und durchschnittlichen Ergebnisse dargestellt. Die halbkomplexen Verfahren schneiden wieder mit durchschnittlich 97,94% am besten ab. Interessant ist, dass auf dem PARSE-Korpus die komplexen Verfahren mit 96,81% durchschnittlich fast so schlecht abschneiden wie die einfachen Verfahren mit 96,60%. Dies liegt auch hier an LibSVM, welches als einziges Verfahren schlechter als die einfache Mehrheitswahl abschnitt.

In Tabelle 6.13 sind die Ergebnisse der Evaluation auf dem PARSE-Korpus ohne Worte ersichtlich. Durch das Fehlen der Worte fallen VFI mit 91,98% und HyperPipes mit 78,39% hinter die einfache Mehrheitswahl mit 93,01%. Im Vergleich zu den Ergebnissen mit Worten kam es zu einigen Änderungen bei den Platzierungen. Während IBk um -0,51 Prozentpunkte schlechter wurde, verbesserten sich J48 (0,65 Prozentpunkte), NNge (0,50 Prozentpunkte) und REPTree (1,15 Prozentpunkte) Das instanzbasierte Verfahren NNge ist nun mit 98,66% das beste Verfahren und übertrifft IBk auf Worten mit 98,57% um 0,09 Prozentpunkte. IBk wurde weiterhin von J48 mit 98,62% und KStar mit 98,48% übertroffen. NNge ist ein instanzbasiertes Verfahren, das Regeln nutzt, was wie bei Entscheidungsbäumen, zu einer Überbewertung der Wörter als Attribut führt. Mehrdeutige Worte werden so nur anhand der Worte klassifiziert, anstatt die Markierungen einzubeziehen. Ist im Trainingskoprus das mehrdeutige Wort nur ein einziges mal enthalten, so wird es immer gleich klassifiziert. Insgesamt fand auf der Hälfte der Verfahren, wie in Tabelle 6.14 ersichtlich, eine Verbesserung statt.

Betrachtet man die Komplexität, so stellt man fest, dass alle fünf einfachen Verfahren eine Verschlechterung erfahren haben. Dies führt zum Sinken der Genauigkeit der einfachen Verfahren auf dem PARSE-Korpus ohne Worte auf durchschnittlich 90,93%. Die komplexen Verfahren hingegen können alle eine Verbesserung durch fehlende Worte erreichen. Die durchschnittliche Genauigkeit dieser Verfahren steigt damit auf 98,08%. Dies entspricht ungefähr den halbkomplexen Verfahren die durchschnittlich 98,07% erreichen.

6.2.4. Fazit

Als Fazit kann festgestellt werden, dass das Gesamtverfahren auf allen drei Korpora die Ergebnisse verbesserte. Hypothese H_1 kann daher angenommen werden. Da die einfache

Tabelle 6.14.: Auswirkung von Worten auf das Ergebnis auf dem PARSE-Korpus

Name	Komplexität	Mit	Ohne	δ in %Punkten
BayesNet	Einfach	97,88%	95,62%	-2,26
HyperPipes	Einfach	94,47%	78,39%	-16,08
IBk	Halbkomplex	98,57%	98,06%	-0,51
J48	Halbkomplex	97,97%	98,62%	0,65
JRip	Komplex	97,88%	98,02%	0,14
KStar	Komplex	98,34%	98,48%	0,14
LibSVM	Komplex	92,95%	95,53%	2,58
NNge	Komplex	98,16%	98,66%	0,50
NaiveBayes	Einfach	97,37%	95,62%	-1,75
OneR	Einfach	95,48%	93,04%	-2,44
PART	Halbkomplex	98,48%	97,70%	-0,78
REPTree	Halbkomplex	96,73%	97,88%	1,15
Ridor	Komplex	96,73%	96,96%	0,23
VFI	Einfach	97,79%	92,98%	-5,81

Mehrheitswahl auf keinem Korpus das beste Verfahren war, kann auch Hypothese H_2 angenommen werden. Auf den meisten Korpora gab es weiterhin ein Verfahren, dass schlechter als der Vergleichsmarkierer war. Hypothese H_3 ist daher zu verwerfen. Die größte Verbesserung fand auf den Korpora statt, auf denen die Wortartmarkierer nicht trainiert wurden. Die Wahl des Verfahrens ist dabei auch von Bedeutung. Nicht jedes Verfahren führt zu einer Verbesserung und es gibt sogar Verfahren, die das Ergebnis verschlechtern. Weiterhin kann festgestellt werden, dass die Qualität des Verfahrens nicht mit der Komplexität des Verfahrens korreliert. Die komplexen Verfahren waren auf keinem Korpus die besten. Auf dem WSJ-Korpus sind die besten drei Verfahren, die die auf Entscheidungsbäumen basieren. Auf den anderen beiden Korpora konnten die instanzbasierten Verfahren die besten Ergebnisse erzielen. Die Entscheidungsbäume folgten dahinter. Dieser Unterschied könnte mit einer Ähnlichkeit der Sätze der Korpora und der unterschiedlichen Funktionsweise der Verfahren erklärt werden. Während Entscheidungsbäume Attribute einzeln iterativ auswerten, werden bei instanzbasierten Verfahren alle Attribute zeitgleich ausgewertet. Es ist daher in Abschnitt C ersichtlich, dass Entscheidungsbäume das Ergebnis des besten Markierers korrigieren, während instanzbasierte Verfahren ein neues Ergebnis klassifizieren. Entscheidungsbäume scheinen daher stärker von einem guten Ergebnis der Wortartmarkierer abzuhängen als instanzbasierte Verfahren. Hypothese H_4 kann daher ebenfalls verworfen werden. Die größte Verbesserung im Vergleich zum Vergleichsmarkierer fand auf Texten statt, die sprachlich weiter von Trainingskorpus der Markierer entfernt sind. Während auf dem WSJ-Korpus eine Verbesserung von 0,41 Prozentpunkten zum Vergleichsmarkierer erreicht wurde, konnte auf den NLCI-Korpus diese auf 2,94 Prozentpunkte erhöht werden. Die größte Verbesserung zum Vergleichsmarkierer gab es auf dem PARSE-Korpus mit 7,91 Prozentpunkten. Die Differenz der Genauigkeit δ zwischen den Korpora verringerte sich, wodurch Hypothese H_5 angenommen werden kann. Auch ist festzustellen, dass auf dem NLCI-Korpus mit 99,11% das beste Ergebnis erzielt wurde. Da bei den Wortartmarkierern das beste Ergebnis auf dem WSJ-Korpus mit 98,54 erreicht wurde, ändert sich die Anordnung der δ zwischen den Korpora. Hypothese H_6 muss daher verworfen werden.

Die Auswirkungen der fehlenden Worte führen je nach Korpus und Verfahren zu unterschiedlichen Ergebnissen. Betrachtet man die durchschnittlichen Genauigkeit verschiedener Verfahren, so ist besonders bei einfachen Verfahren der größte Verlust der Genauigkeit zu bemerken. Zwar ist für diesen Verlust oft HyperPipes verantwortlich, aber auch die an-

deren Verfahren fallen fast immer in ihrer Genauigkeit. Einzig das NaiveBayes Verfahren kann auf dem WSJ-Korpus, durch die fehlenden Worte eine Verbesserung von 0,05 Prozentpunkten erfahren. Bei komplexen Verfahren hingegen lässt sich öfters feststellen, dass diese ohne Worte besser funktionieren. Aber auch hier ist dies vom Korpus abhängig. Während auf dem PARSE-Korpus alle komplexen Verfahren ohne Worte besser abschneiden, verlieren die meisten komplexen Verfahren auf dem NLCI-Korpus an Genauigkeit. Auch wenn man von den durchschnittlichen Genauigkeiten abweicht und nur die besten Verfahren beachtet, so kann die Hypothese nicht bestätigt werden. Auf dem PARSE-Korpus konnte NNge ein besseres Ergebnis ohne Worte erzielen, als IBk mit Worten. Hypothese H_7 lässt sich deshalb nicht bestätigen und muss ebenfalls verworfen werden.

Hypothese H_8 kann hingegen angenommen werden. Auf allen Korpora konnte auch ohne Verwendung von Worten mindestens ein Verfahren festgestellt werden, dass den Vergleichsmarkierer und die einfache Mehrheitswahl übertreffen konnte. Abhängig vom Korpus unterscheiden sich diese Verfahren aber. Während auf dem WSJ-Korpus J48 und auf dem NLCI-Korpus IBk unabhängig der Worte, jeweils die besten Verfahren waren, unterschieden sich die besten Verfahren auf dem PARSE-Korpus. Bei Verwendung von Worte konnte sich IBk gegenüber den restlichen Verfahren durchsetzen. Werden die Worte weggelassen, so fällt IBk und NNge steigt auf den ersten Platz auf.

6.3. Monte-Carlo-Kreuzvalidierung

Während bei der 10-fachen Kreuzvalidierung nur festgestellt werden konnte, ob die Ergebnisse optimiert wurden, so kann über die Monte-Carlo-Kreuzvalidierung die Größe des Trainingskorpus, der für eine Verbesserung nötig ist, untersucht werden. Hierzu wird die Datenmenge, abhängig vom festgelegten Trainingsanteil, in zwei zufällige Mengen aufgeteilt. Die Größe der Trainingsmenge bleibt dabei immer gleich. Auf der einen Menge wird ein Modell trainiert, das auf der anderen getestet wird. Dies wird mehrmals wiederholt.

Durch diese Methode kann die Größe des Trainingskorpus durch Anpassen der Anteilsverhältnisse verändert werden. Um zu untersuchen welche Größe ausreichend ist, wird mit verschieden Anteilsverhältnissen evaluiert. Da die größten Änderungen im niedrigen Prozentbereich stattfinden, soll dieser Bereich etwas feiner aufgeteilt werden. Damit kann festgestellt werden, ab welchen Anteilsverhältnissen der Trainingskorpus für eine Verbesserung ausreichend ist.

Die Ergebnisse der Evaluation werden graphisch dargestellt. Dabei entspricht die horizontale Achse den Anteilsverhältnissen, während die vertikale Achse den Erwartungswert der Genauigkeit abbildet. Eine Linie durch die Erwartungswertachse zeigt die Genauigkeit des jeweiligen Vergleichsmarkierer an. Da es Aufgrund der Vielzahl an untersuchten Verfahren schwer ist, diese alle in einer einzelnen Grafik darzustellen, werden jeweils nur sieben Verfahren pro Grafik abhängig vom Korpus dargestellt. Da die Lernkurven sich teilweise stark in den Bereichen unter 10% und über 10% unterscheiden, werden diese Bereiche fokussiert dargestellt. Weiterhin werden für die Lernkurven der Verfahren, die keine Worte als Attribut nutzen, eigene Grafiken abgebildet. Dadurch gibt es für jeden Korpus insgesamt zwölf Abbildungen. Die Lernkurven und dazu gehörige Sättigungstabellen finden sich aus Gründen der Lesbarkeit am Ende des Kapitels. In den Sättigungstabellen steht das Wachstum der Lernkurven abhängig von den Anteilsverhältnissen und der Anzahl Worte. Anhand dieser Tabellen kann einfach nachgeschaut werden, ab welcher Trainingsgröße die Verfahren kaum noch Verbesserung zeigen. Eine tabellarische Darstellung der Lernkurven findet sich in Abschnitt B

6.3.1. WSJ-Korpus

In Abbildung 6.1 und Abbildung 6.2 sind die Ergebnisse der Monte-Carlo-Kreuzvalidierung für den WSJ-Korpus dargestellt. Die Graphen lassen sich leicht in zwei Bereiche teilen. Der Bereich für Trainingsanteile unter 10% ist in Abbildung 6.3 und Abbildung 6.4 fokussiert abgebildet. Da der Korpus stark unterschiedlich scheint, würde die geringe Trainingsmenge zu starken Variationen im Ergebnis führen. Eine Besonderheit stellt HyperPipes dar. Dieses Verfahren hat in diesem Bereich seine höchste Genauigkeit und sinkt mit mehr Trainingsdaten. Aber HyperPipes und kein anderes Verfahren kann mit einer Trainingsmenge von unter 10% den Vergleichsmarkierer übertrumpfen. Wird die Trainingsmenge in Abbildung 6.5 und Abbildung 6.6 auf mehr als 10% erhöht, so schafft es J48 bereits bei 10% den Vergleichsmarkierer mit 98,60% zu optimieren. Eine weitere Erhöhung der Trainingsmenge führt zu geringen Verbesserungen. Bis auf HyperPipes führt eine Erhöhung der Trainingsmenge bei allen Verfahren zu Verbesserungen im Ergebnis. Die Wirkung sinkt aber je höher der Anteil der Trainingsmenge ist. Eine Erhöhung von 10% auf 20% führt zu einer stärkeren Verbesserung, als eine Erhöhung von 80% auf 90%. Es findet eine Sättigung der Verfahren statt. Einzig bei J48 scheint keine Sättigung stattzufinden, da zwischen 70% und 90% eine ähnlich große Verbesserung eintritt wie zwischen 10% und 70%. Eine mögliche Erklärung wäre, dass ab 70% Zweige für spezielle Ausnahmefälle erzeugt werden.

Die Graphen in Abbildung 6.7 und Abbildung 6.8 zeigen die Lernkurven der Verfahren abhängig vom Trainingsanteil. Besonders die Lernkurve von HyperPipes und VFI stechen heraus, da sie mit erhöhten Trainingsanteil sinken. Eine Idee um dieses Verhalten von HyperPipes zu erklären, könnte sein, dass das Verfahren durch mehr Trainingsdaten überlastet ist. Während dem Training erstellt HyperPipes bei jeder Klasse für jedes Attribut eine Untermenge von Werten. Diese Untermenge sagt aus, dass ein bestimmter Wert des Attributes bei dieser Klasse vorkommt. Da hierbei nur die Existenz des Attributwerts von Bedeutung ist, wirken sich Ausreißer genauso stark aus, wie der Normalfall. Um eine Instanz zu klassifizieren, überprüft HyperPipes ob die Werte der Instanz Teil der Untermenge für die Klasse sind. Die Klasse, in der die meisten Werte der Attribute liegen, wird ausgewählt. Mehr Trainingsdaten führen dazu, das bei einigen Klassen (NN, NNP, NNS, VBZ, JJ, etc.) die Untermenge alle Werte abdeckt. Das Verfahren ist damit besonders ohne Worte überlastet. Werden Worte verwendet, so dienen diese als starkes Unterscheidungsmerkmal, wenn die Untermengen bei den Markierungen zu allgemein werden. Die Lernkurven der anderen Verfahren steigen hingegen alle, bis eine Sättigung eintritt. Dies ist im Bereich für Anteilsverhältnisse unter 10% in Abbildung 6.9 und Abbildung 6.10 leicht ersichtlich. Im Bereich für Anteilsverhältnisse über 10% in Abbildung 6.11 und Abbildung 6.12 zeigt sich ein ähnliches Bild. Bis auf HyperPipes und VFI steigen alle Verfahren auch dort. Die Lernkurven sind aber nicht monoton steigend. Eine Ursache hierfür könnte die Varianz bei der Auswahl der Trainingsmenge sein. Da nur zehn Iterationen durchgeführt wurden, führt die zufällige Wahl von besonders schlechten oder guten Trainingsinstanzen dazu, dass die Kurve in diese Richtung ausschlägt. Dem kann durch eine Erhöhung der Anzahl an Iterationen bei der Monte-Carlo-Kreuzvalidierung entgegengewirkt werden.

Unabhängig, ob Worte verwendet werden, wird ein Trainingsanteil von 10% benötigt, damit J48 eine Verbesserung erreichen kann. In Tabelle 6.15 ist die Sättigungstabelle für den WSJ-Korpus unter Verwendung von Worten abgebildet. Die Sättigungstabelle für den WSJ-Korpus mit Worten findet sich in Tabelle 6.16. Die Tabellen zeigen das Wachstum der Lernkurven. Dabei findet, wie auch in den Graphen ersichtlich, eine Sättigung statt, die abhängig vom Verfahren unterschiedlich schnell eintritt. Bei der Mehrheit der Verfahren führt eine Erhöhung des Trainingsanteils um 10 Prozentpunkte ab 30% nur noch zu einer Verbesserung von unter 0,10 Prozentpunkten.

6.3.2. NLCI-Korpus

In Abbildung 6.13 und Abbildung 6.14 sind die Graphen für den NLCI-Korpus abgebildet. Wie in Abbildung 6.15, Abbildung 6.16, Abbildung 6.17 und Abbildung 6.18 genauer dargestellt, führt eine Erhöhung der Trainingsmenge dazu, dass die Genauigkeit steigt. Eine Begründung hierfür könnte die stärkere Ähnlichkeit der Sätze im Vergleich zum WSJ-Korpus sein. Damit beeinflusst, besonders bei geringen Anteilsverhältnissen, die Wahl der zufälligen Trainingsmenge das Ergebnis weniger.

Die Lernkurven auf dem NLCI-Korpus ohne Worte sind in Abbildung 6.19 und Abbildung 6.20 dargestellt. Wie auch auf dem WSJ-Korpus ohne Worte, scheint HyperPipes überlastet zu sein und fällt mit mehr Trainingsanteil. Da dieser Korpus aber weniger Worte enthält, kann sich diese Überlastung nicht so stark auswirken. Die Untermengen der Klassen scheinen genug Raum zum Differenzieren zu lassen. Die anderen Verfahren können hingegen mit mehr Trainingsdaten bessere Ergebnisse liefern. In Abbildung 6.21 und Abbildung 6.22 sind die Lernkurven für die Anteilsverhältnisse unter 10% dargestellt. Die Anteilsverhältnisse über 10% sind in Abbildung 6.23 und Abbildung 6.24 abgebildet.

Wie auch mit Worten, reicht ein Trainingsanteil von 2% aus, um den Vergleichsmarkierer zu übertreffen. Dies ist erstaunlich wenig. Neben der niedrigeren Schwelle durch den Wortartmarkierer, wäre wohl auch die stärkere Ähnlichkeit des Korpus eine Begründung. Die beiden Sättigungstabellen für den NLCI-Korpus sind in Tabelle 6.17 und Tabelle 6.18 dargestellt. Daraus ist ersichtlich, dass eine Verbesserung von 30% auf 40% bei den meisten nur noch zu einer Verbesserung von unter 0,20 Prozentpunkten führt.

6.3.3. PARSE-Korpus

Bis auf HyperPipes, konnte eine Verbesserung der Ergebnisse bei Erhöhung der Trainingsmenge bei allen Verfahren in Abbildung 6.25 und Abbildung 6.26 festgestellt werden. In Abbildung 6.27 und Abbildung 6.28 ist ersichtlich, dass bereits ein Trainingsanteil von 5% ausreicht, um die Ergebnisse zu verbessern. Die Lernkurven für den Bereich von über 10% in Abbildung 6.29 und Abbildung 6.30 zeigen, dass die Kurven mit mehr Trainingsdaten meist steigen. Dabei ist bereits bei geringen Trainingsanteilen festzustellen, welche Verfahren bei großen Trainingsanteilen besser sind. Die Ausnahme bildet hierbei HyperPipes, da es mit mehr Trainingsdaten sinkt.

Auch auf dem PARSE-Korpus ohne Worte in Abbildung 6.31 und Abbildung 6.32 zeigt sich ein ähnliches Bild. Dies ist in Abbildung 6.33 und Abbildung 6.34 für den Bereich unter 10% Trainingsanteil vergrößert dargestellt. Der Bereich über 10% ist in Abbildung 6.35 und Abbildung 6.36 ersichtlich. HyperPipes fällt wie auf allen Korpora ohne Worte. Die Lernkurve von VFI steigt bis die höchste Genauigkeit bei 40% Trainingsanteil erreicht wird, um danach zu fallen. VFI scheint wie HyperPipes mit mehr Trainingsdaten ebenfalls überlastet zu sein. Bei den restlichen Verfahren führen mehr Trainingsdaten zu einem besseren Ergebnis.

Um auf dem PARSE-Korpus eine Verbesserung zu erreichen, wird ein Trainingsanteil von 5% benötigt. Dies entspricht 109 Worte bei einer Korpusgröße von 2173 Worten. In Tabelle 6.19 und Tabelle 6.20 sind die Sättigungstabellen für den PARSE-Korpus dargestellt. Da auch bei hohen Anteilsverhältnissen noch akzeptable Steigerungen vorhanden sind, ist es schwer auf diesem Korpus einen passende Sättigung festzustellen. Aber bei Anteilsverhältnissen von über 40% findet bei den meisten Verfahren eine Verbesserung von unter 0,50 Prozentpunkten statt.

6.3.4. Fazit

Bei allen drei Korpora mit Worten konnte im Allgemeinen eine Verbesserung des Ergebnisse durch Erhöhung der Trainingmenge festgestellt werden. Besonders auf kleinen Korpora

mit ähnlichem Satzaufbau führen die Verfahren bereits mit geringen Trainingsmengen zu einer Verbesserung. Aber auch auf dem WSJ-Korpus konnte mit 10% Trainingsanteil eine Verbesserung von 0,06% erreicht werden. Eine weitere Erhöhung der Trainingsmenge führt zu einer Sättigung. Aufgrund der unterschiedlichen Form der Korpora ist es nicht möglich einen Richtwert für die benötigte Wortanzahl festzustellen. Es scheint aber, dass ab 30% Trainingsanteil bei vielen Verfahren nur noch geringe Verbesserungen möglich sind.

Auch auf den Korpora ohne Worte gibt es Verfahren wie HyperPipes und VFI, die durch mehr Trainingsdaten ein schlechteres Ergebnis zeigen. Auf den meisten Verfahren kann aber ein positiver Zusammenhang zwischen Menge der Trainingsdaten und dem Ergebnis festgestellt werden. Da es einige Verfahren gibt, die durch eine Erhöhung der Trainingsmenge schlechter werden, kann Hypothese H_9 nicht angenommen werden. Ursachen hierfür dürften von der Funktionsweise des Verfahrens abhängig sein.

6.4. Zusammenfassung

Anhand der Evaluation der Verfahren lässt sich feststellen, dass es auf jedem Korpus mindestens ein Klassifikationsverfahren gab, dass die Wortartmarkierer erfolgreich kombinieren konnte. Die Verbesserung durch die Kombination hängt hierbei vom Verfahren und dem Korpus ab. Auf den WSJ-Korpus, bei dem die Wortartmarkierer durchschnittlich die höchsten Genauigkeiten erreichten, waren die Entscheidungsbäume auf den besten Plätzen. Auch war der WSJ-Korpus, der einzige Korpus, bei welchem die einfache Mehrheitswahl aller Wortartmarkierer den Vergleichsmarkierer nicht übertreffen konnte. Bei den anderen beiden Korpora dominierten meist die instanzbasierten Klassifikationsverfahren. Die ursprüngliche Genauigkeit der verwendeten Wortartmarkierer scheint zwar Einfluss darauf zu haben, welche Verfahren erfolgreicher sind, aber kaum auf die Qualität des besten Verfahrens. Sind die Wortartmarkierer von hoher Qualität, so schneiden Verfahren wie J48 gut ab, die die Ergebnisse des besten Markierers übernehmen können und nur Korrekturen durchführen. Bei schlechten Wortartmarkierern liegen Verfahren wie IBk vorne, die die Markierung komplett neu klassifizieren. Auf allen Korpora konnten die besten Verfahren über 98,54% erreichen, was die beste Genauigkeit des besten Wortartmarkierers unabhängig vom Korpus darstellt. Die einfache Mehrheitswahl konnte diese Genauigkeit nie erreichen. Da die Wortartmarkierer unterschiedlich gut auf den verschiedenen Korpora abschnitten, führte das Verfahren dazu, dass sich die Unterschiede in der Genauigkeit verringerten.

Bei der Wahl der Verfahren kann festgestellt werden, dass einfache Verfahren kaum geeignet sind. Sie schneiden alle gegenüber den halbkomplexen und komplexen Verfahren schlechter ab und könnten auf dem WSJ-Korpus nicht den Vergleichsmarkierer übertreffen. Die Nutzung von besonders komplexen Verfahren scheint aber auch nicht zum besten Ergebnis zu führen. Zwar waren die komplexen Verfahren besser als die einfachen, konnten sich aber meist gegenüber den halbkomplexen Verfahren auch nicht durchsetzen. Diese waren auf allen drei Korpora die Erfolgreichsten.

Weiterhin konnte festgestellt werden, dass die Auswirkung der Worte als Attribut, nicht vorhersagbar scheint. Während die meisten Verfahren, durch fehlende Worte an Genauigkeit einbüßten, gab es trotzdem Verfahren die dadurch verbessert wurden. Aber auch ohne Worte gab es auf jedem Korpus, mindestens ein Verfahren, dass besser als der Vergleichsmarkierer oder die einfache Mehrheitswahl war.

Für eine Verbesserung ist aber meist nur eine kleine Menge des Korpus als Trainingsmenge nötig. Diese hängt aber vom Korpus ab. Während auf dem PARSE-Korpus bereits 5% von knapp 2174 Worten (ca. 109 Worte) als Trainingsmenge ausreicht, um eine Verbesserung zu erlangen, wird auf dem WSJ-Korpus 10% oder vorzugsweise mehr von ungefähr 94085

Worten (ca. 9409 Worte) benötigt. Vorteilhaft ist aber, dass unabhängig vom Korpus die Verfahren sich gewöhnlicherweise schnell differenzieren. Die Verfahren, die bereits bei geringen Trainingsgrößen Verbesserungen bringen, sind auch bei großen Trainingsmengen meist die besten. Aber es gibt auch Verfahren wie HyperPipes und VFi, die bei einer Erhöhung der Trainingsmenge an Qualität verlieren.

Zusammenfassend lässt sich somit sagen, dass es möglich ist Wortartmarkierer durch Klassifikationsverfahren so zu kombinieren, dass eine Verbesserung stattfindet. Das beste Verfahren hängt dabei aber von einer Vielzahl von Faktoren ab. J48 führte hierbei generell zur stärksten und sichersten Verbesserung. Es war auf dem WSJ-Korpus das beste Verfahren und führte auch auf den anderen Korpora zu einer guten Verbesserung.

Abbildung 6.1.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten (Teil 1)

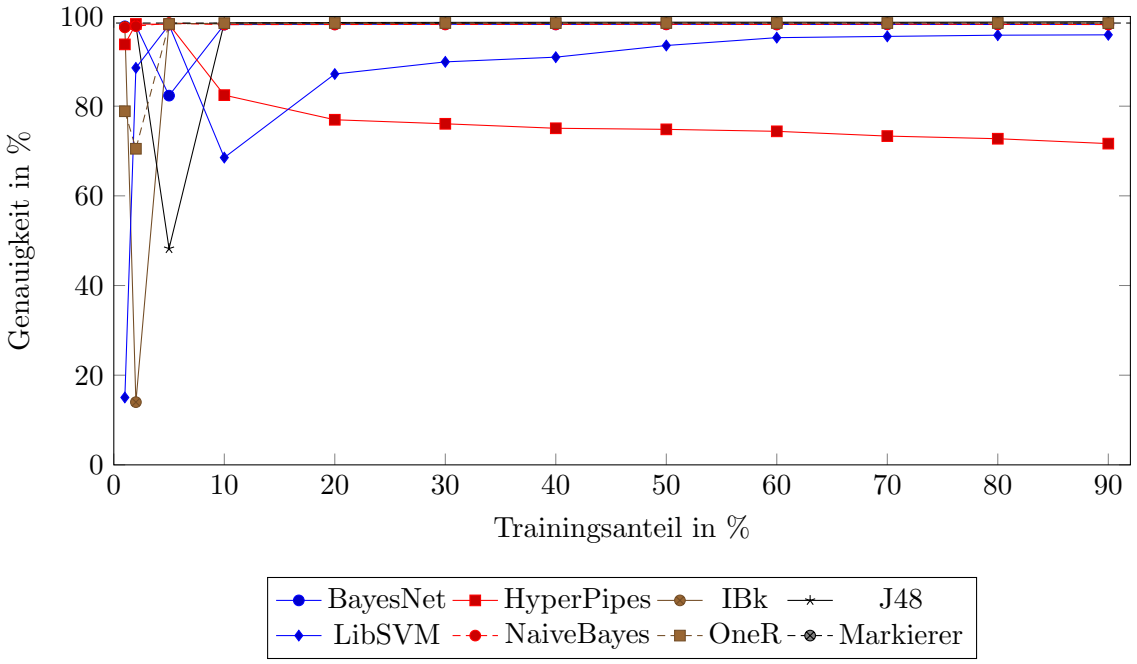

Abbildung 6.2.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten (Teil 2)

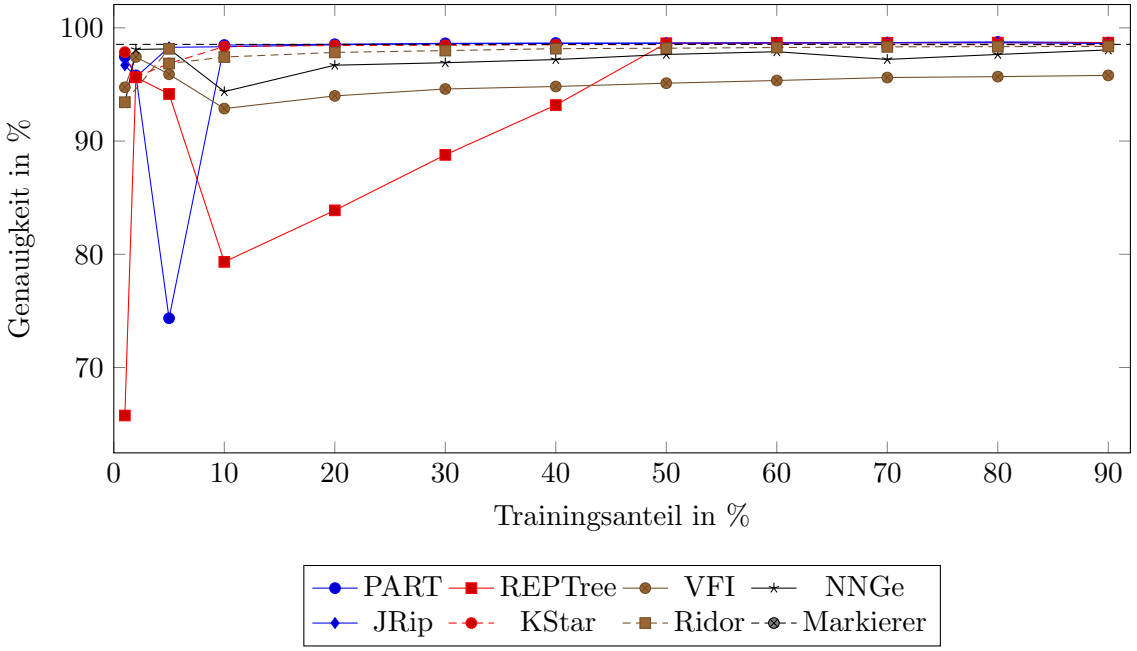

Abbildung 6.3.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 1)

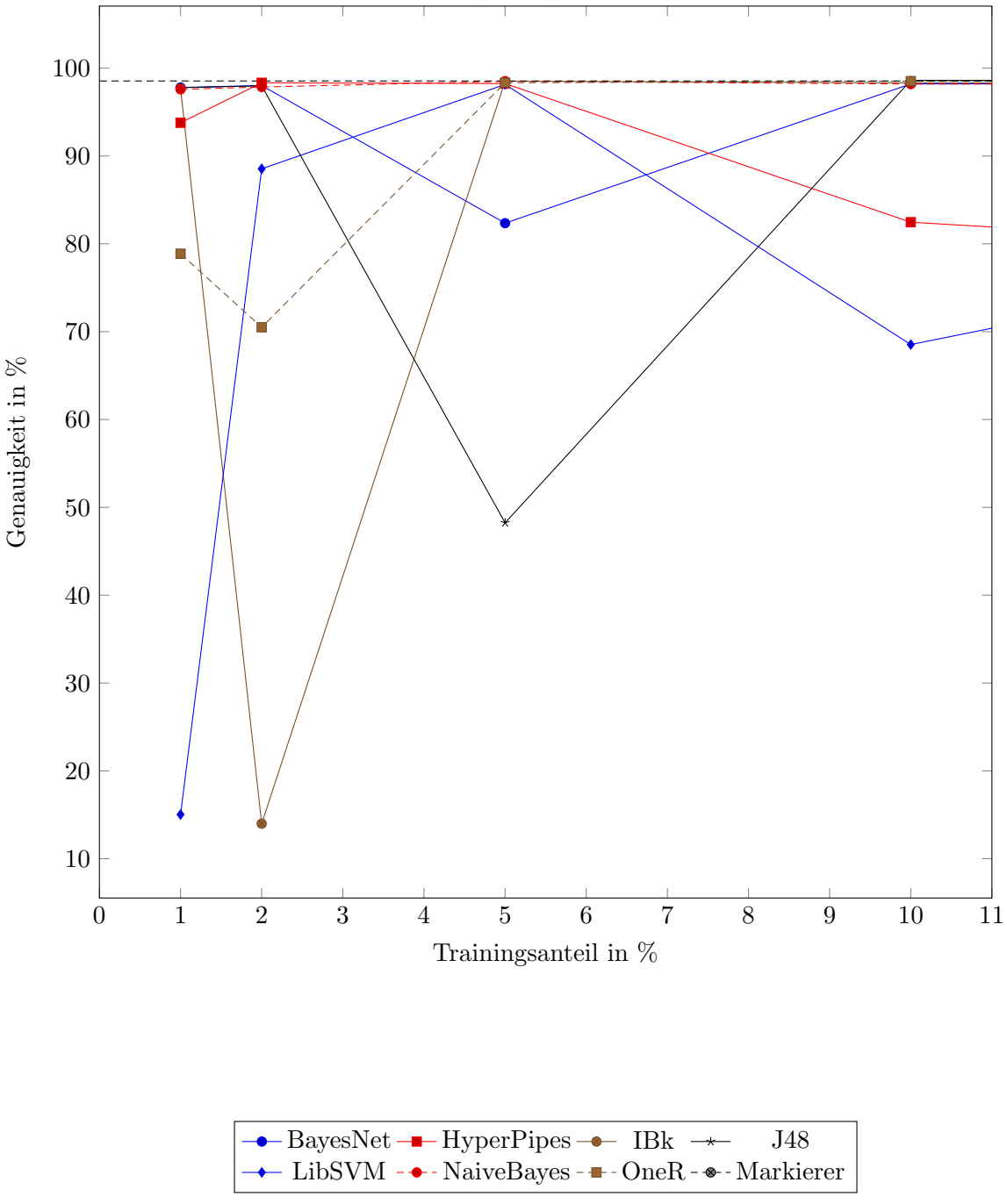

Abbildung 6.4.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 2)

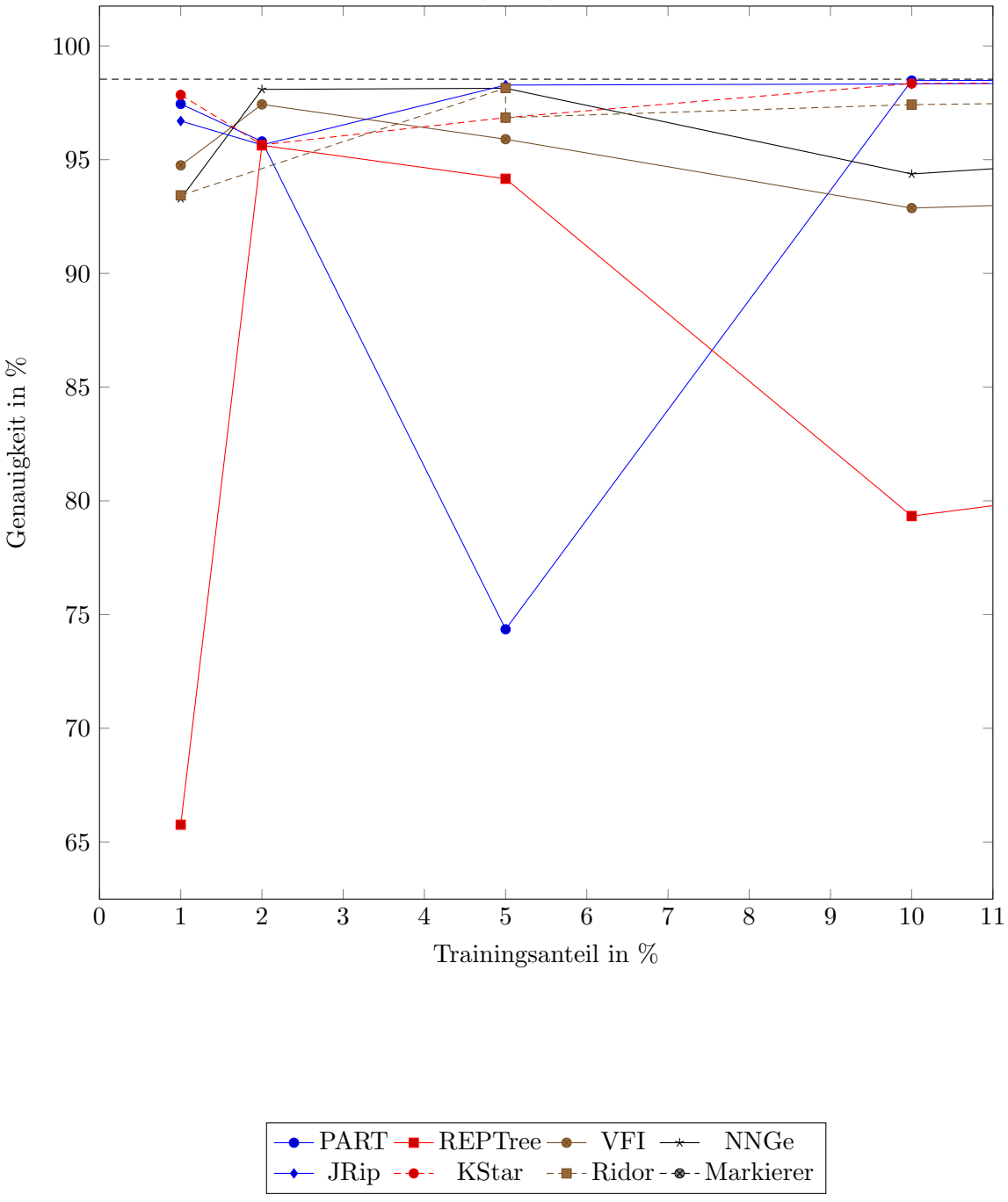

Abbildung 6.5.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten bei Trainingsanteil über 10% (Teil 1)

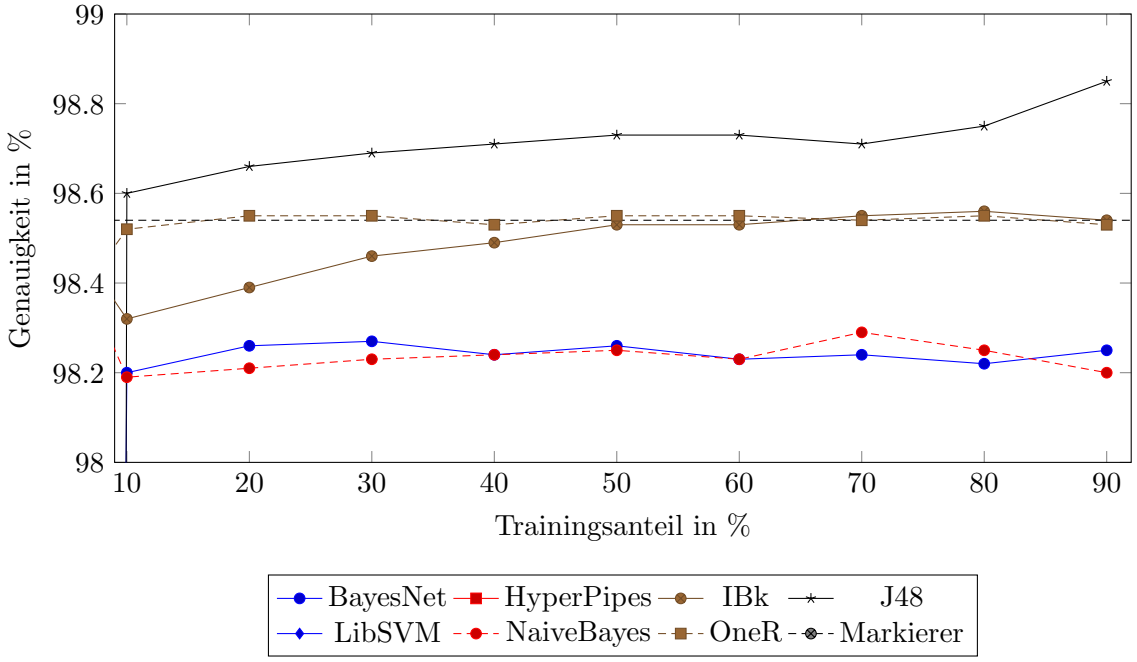

Abbildung 6.6.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus mit Worten bei Trainingsanteil über 10% (Teil 2)

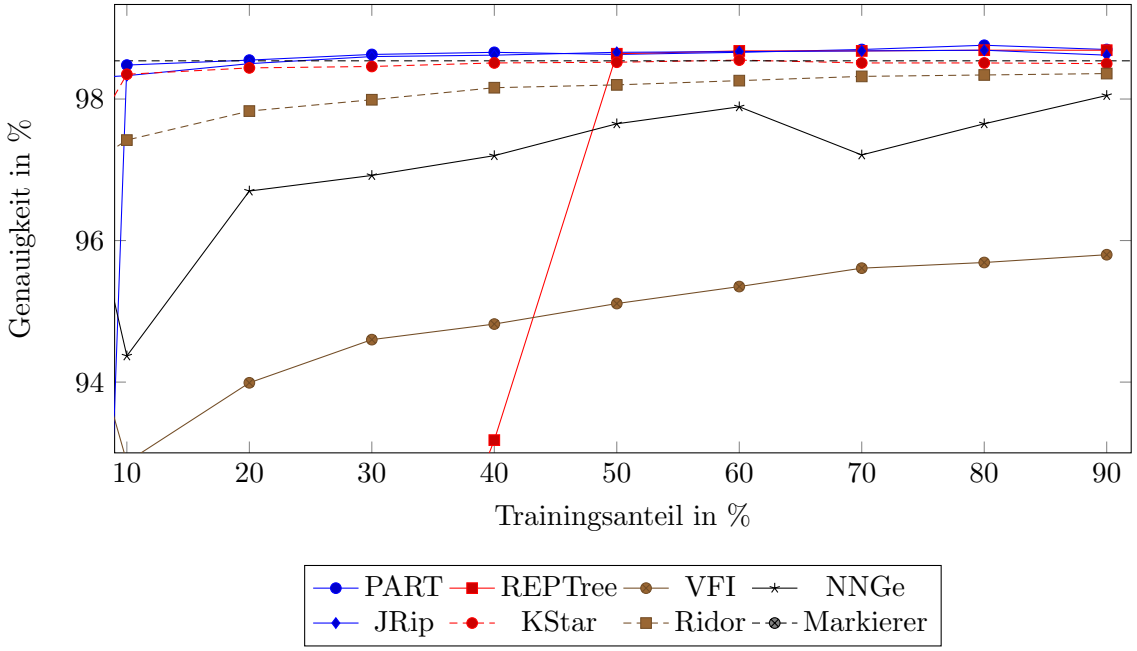

Abbildung 6.7.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte (Teil 1)

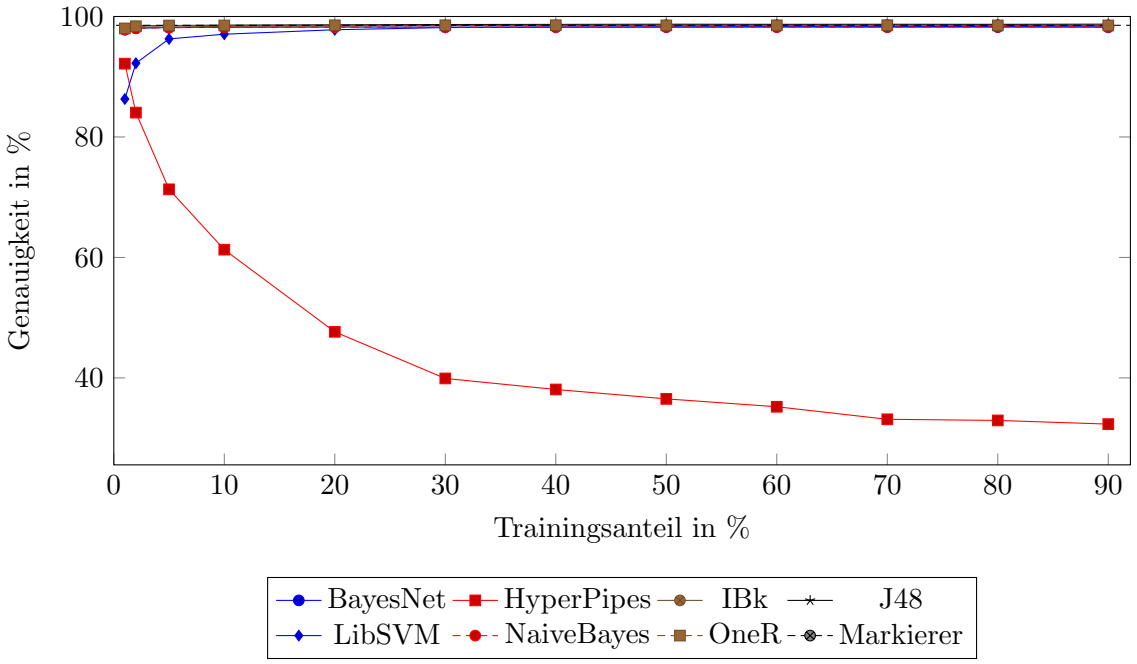

Abbildung 6.8.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte (Teil 2)

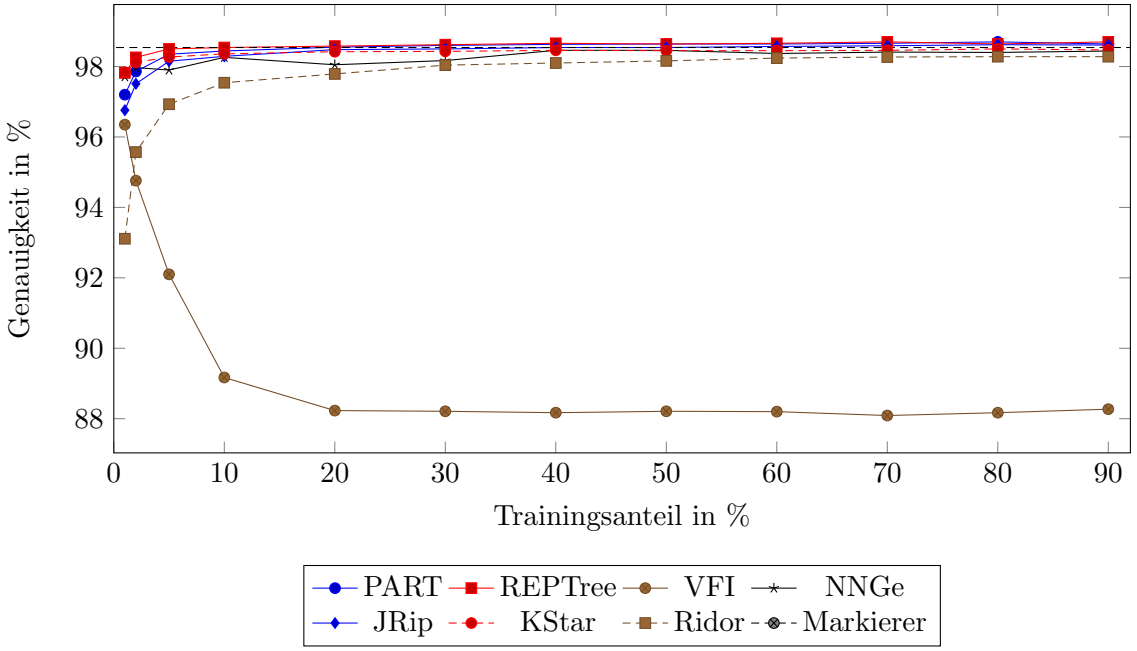

Abbildung 6.9.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 1)

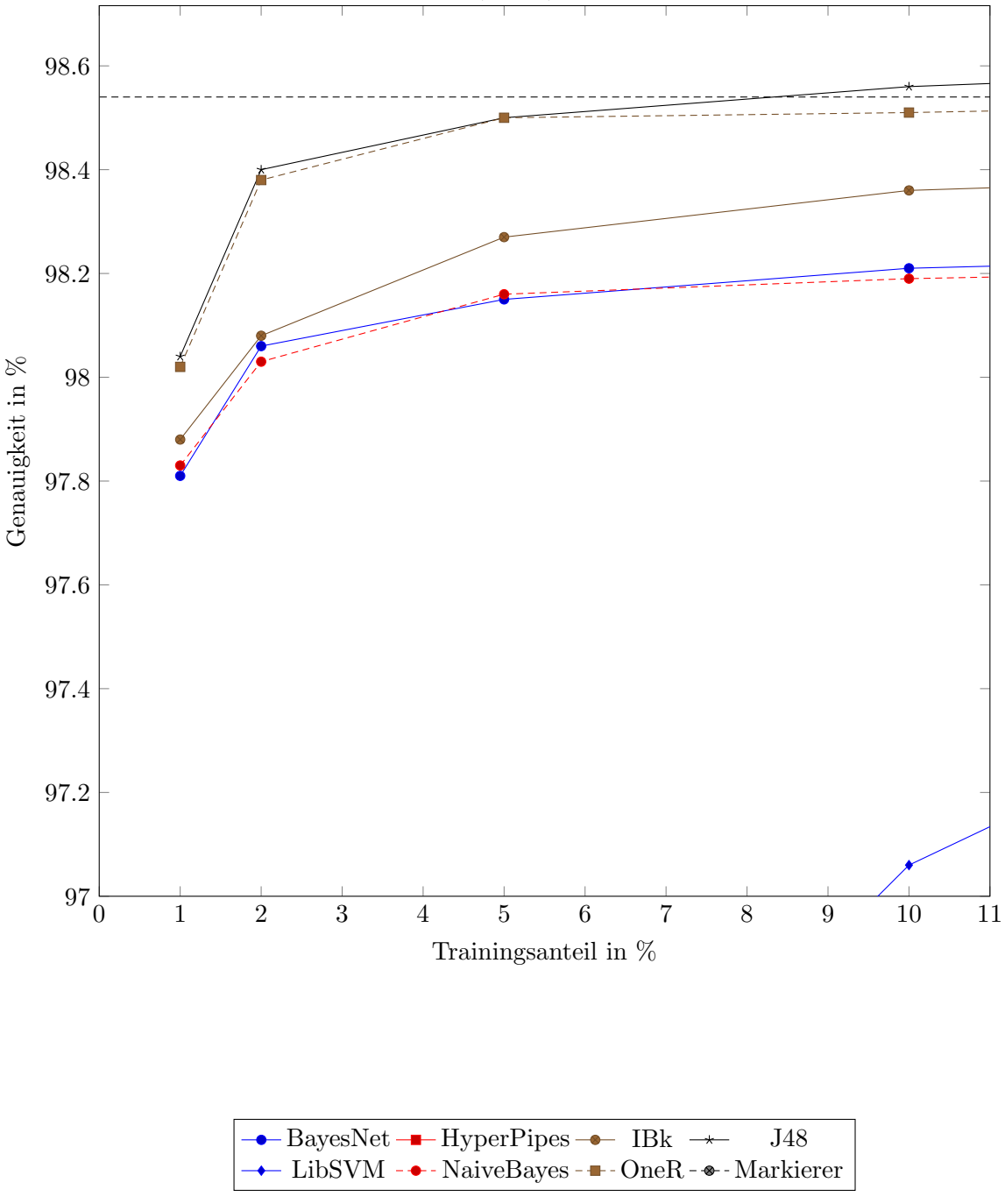

Abbildung 6.10.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 2)

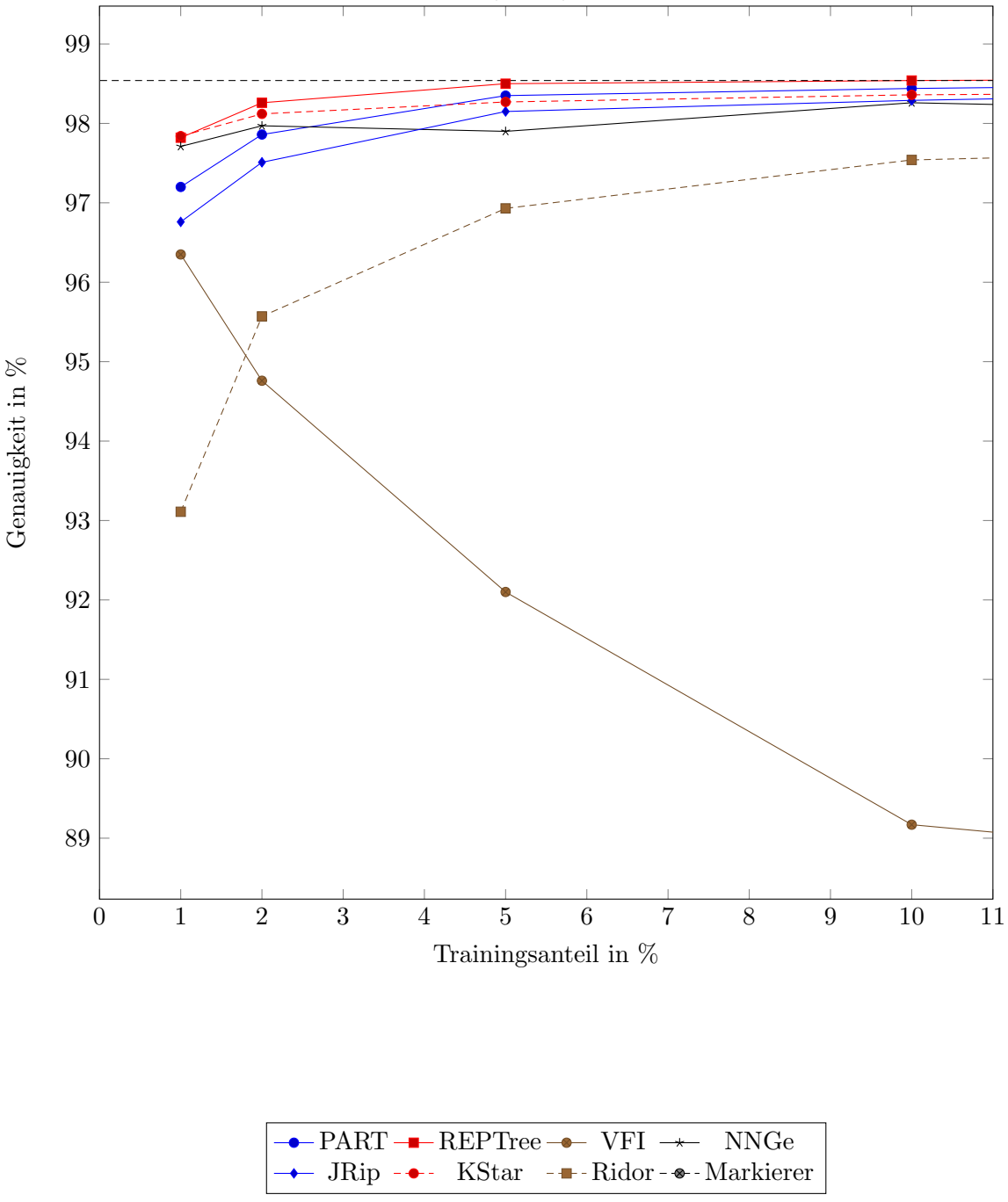

Abbildung 6.11.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte mit Trainingsanteil über 10% (Teil 1)

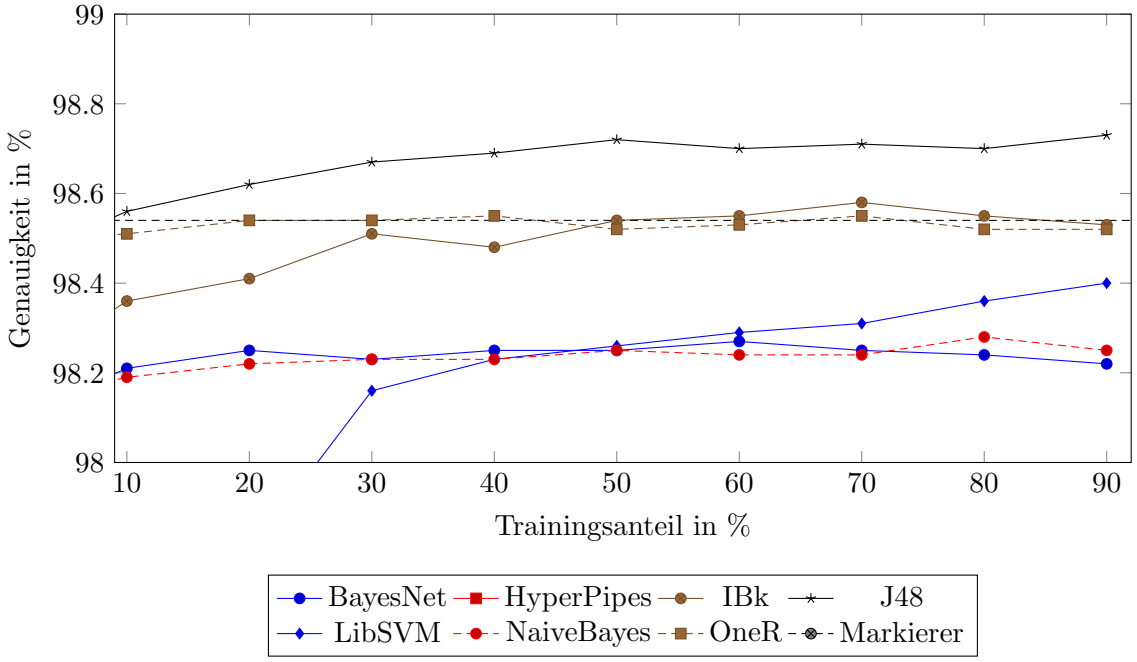

Abbildung 6.12.: Lernkurven der Klassifikationsverfahren auf WSJ-Korpus ohne Worte mit Trainingsanteil über 10% (Teil 2)

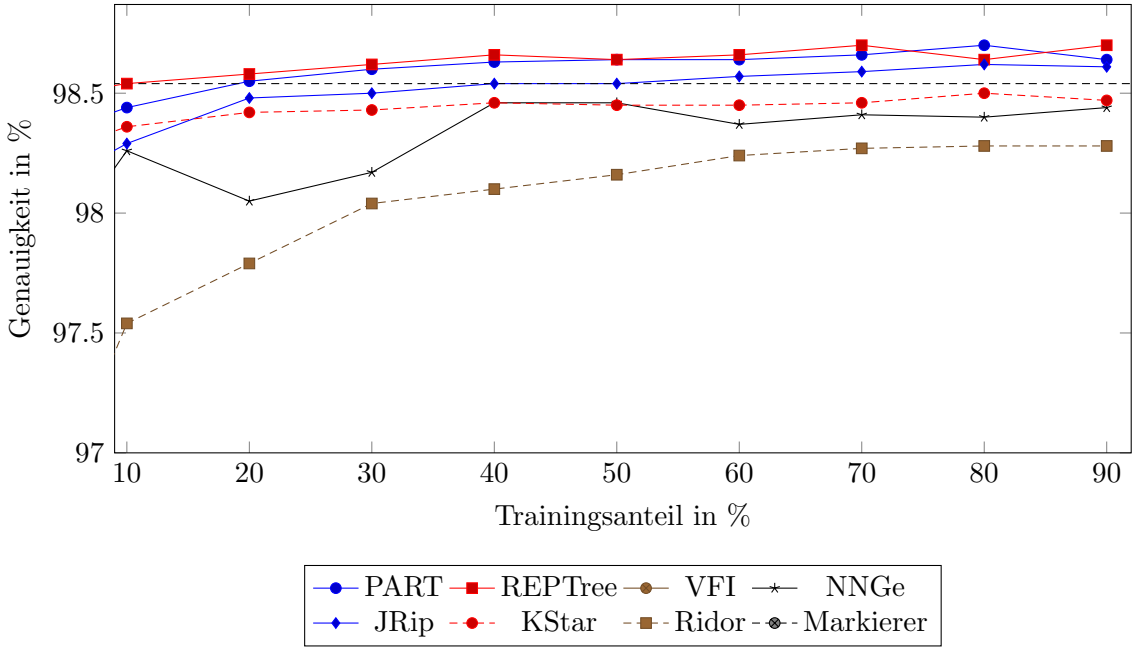

Abbildung 6.13.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten (Teil 1)

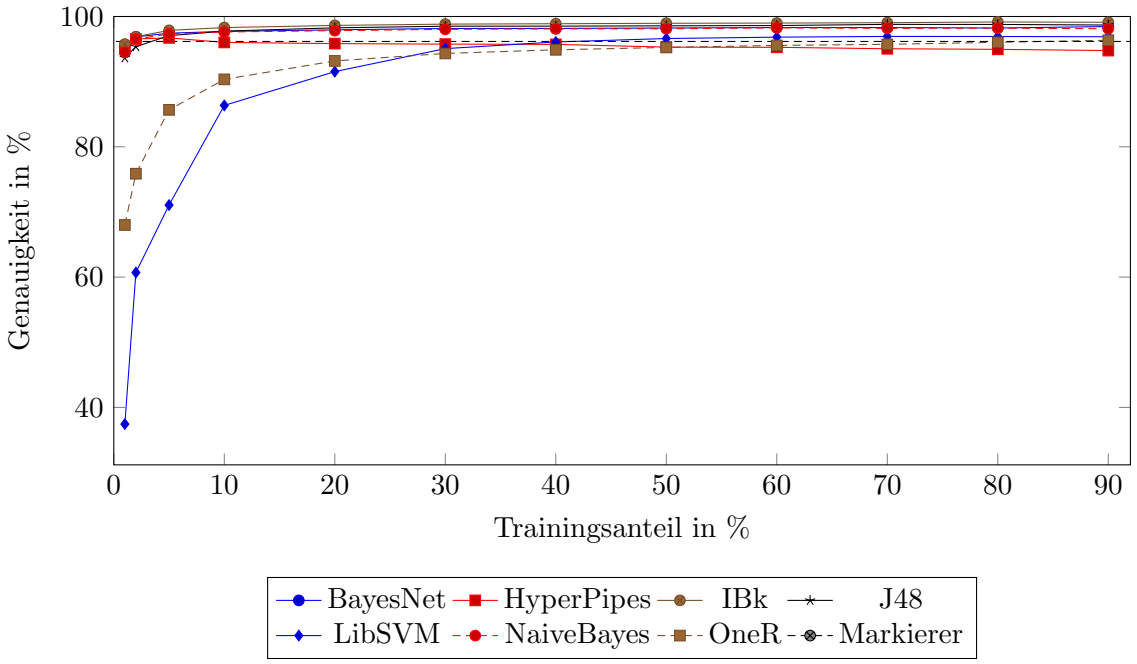

Abbildung 6.14.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten (Teil 2)

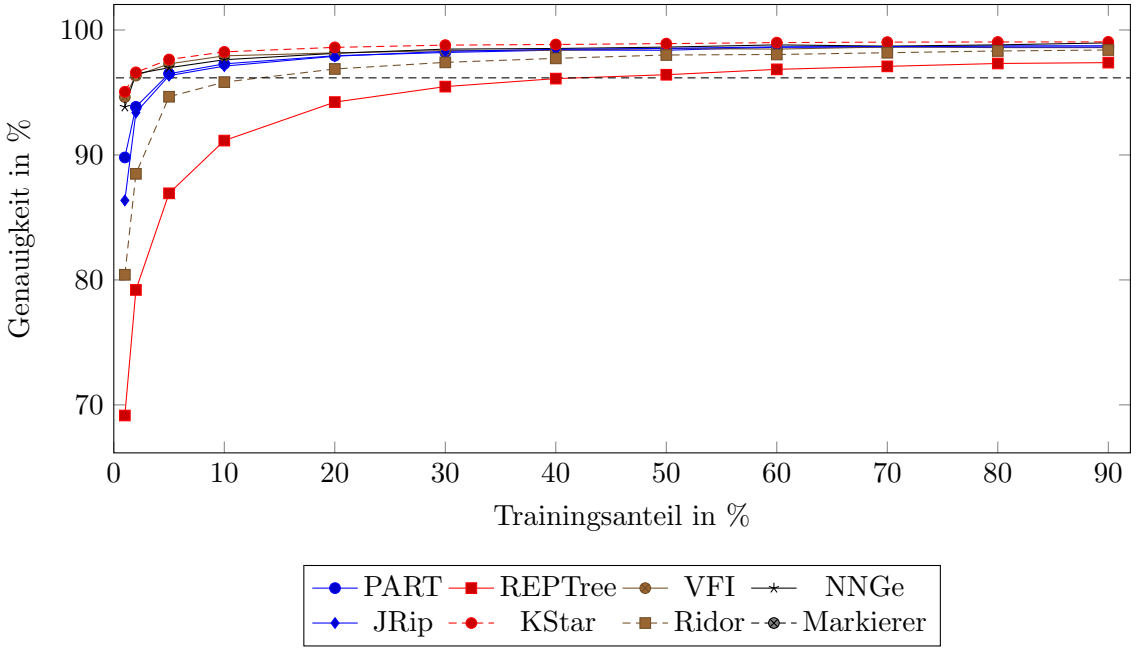

Abbildung 6.15.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 1)

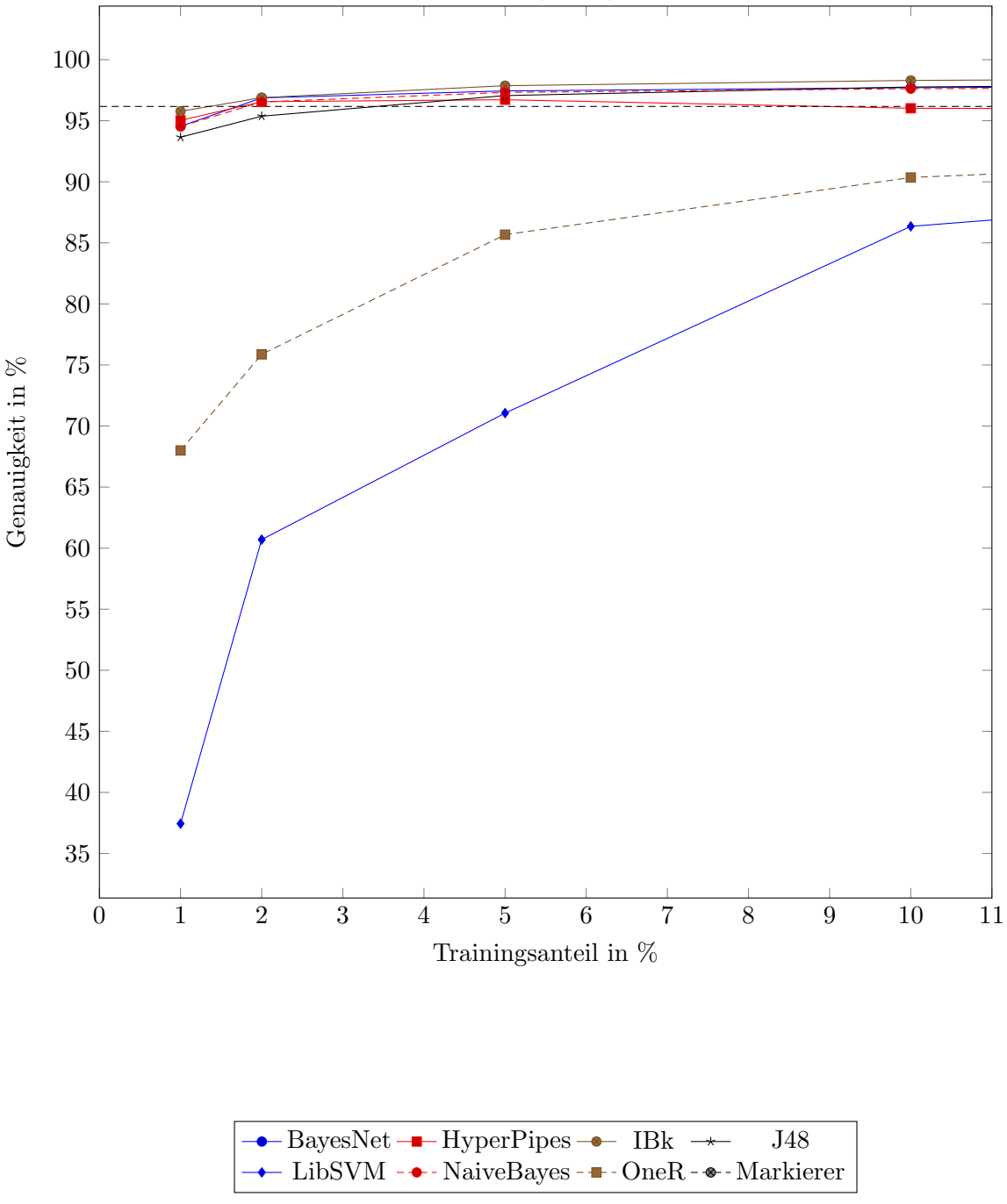

Abbildung 6.16.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 2)

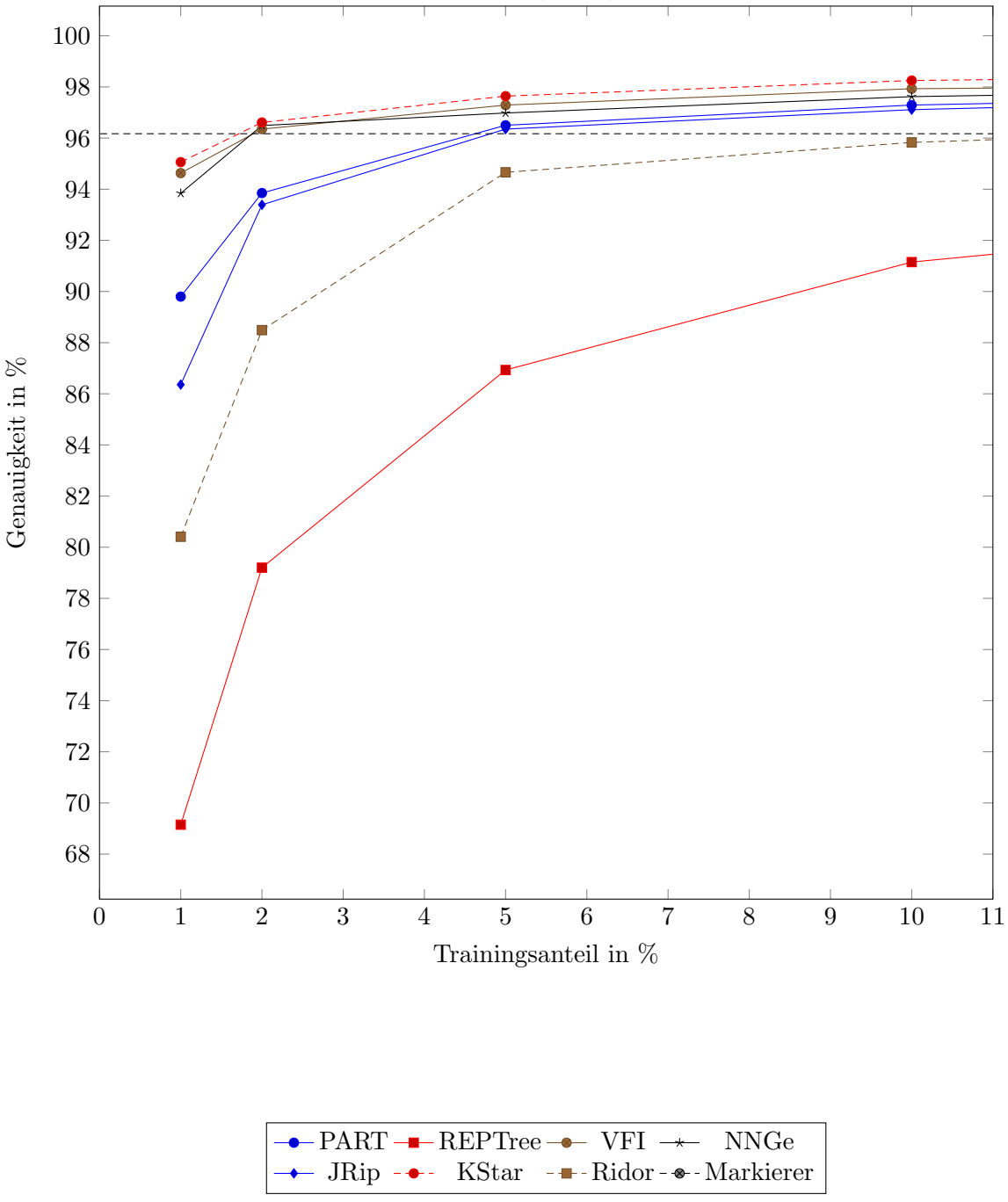

Abbildung 6.17.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten bei Trainingsanteil über 10% (Teil 1)

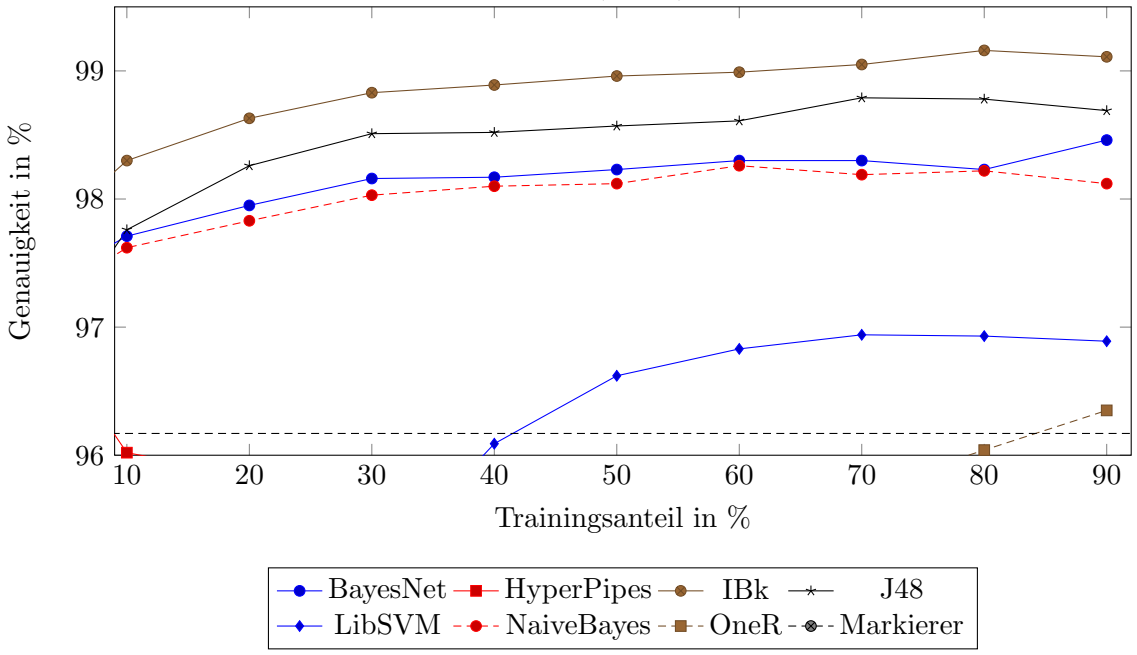

Abbildung 6.18.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus mit Worten bei Trainingsanteil über 10% (Teil 2)

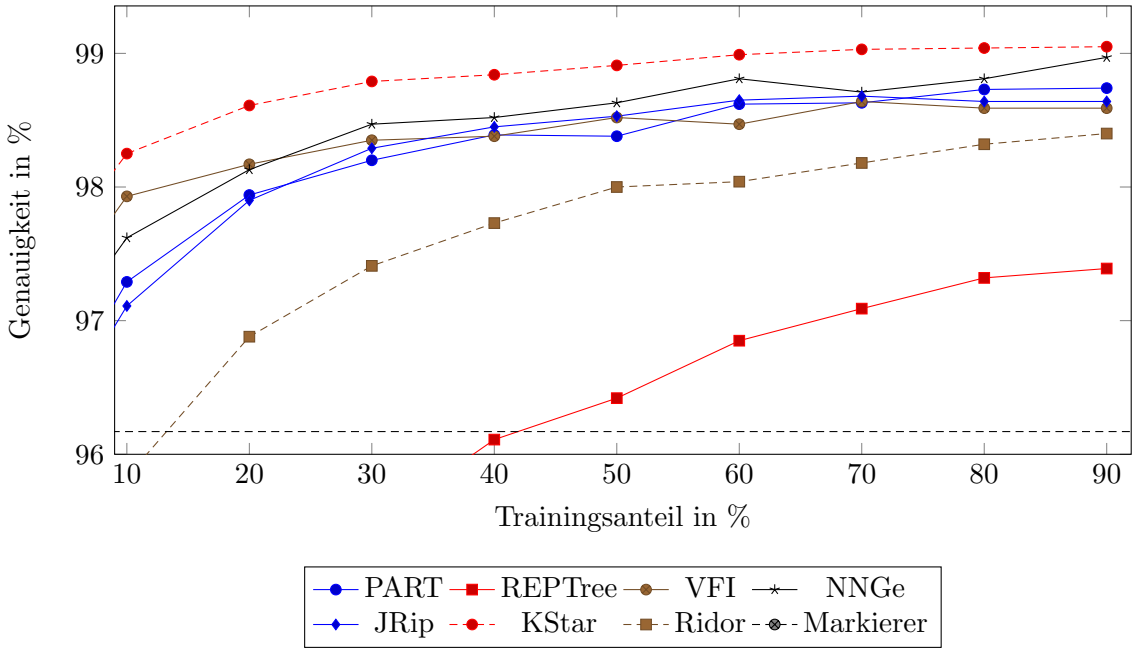

Abbildung 6.19.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte (Teil 1)

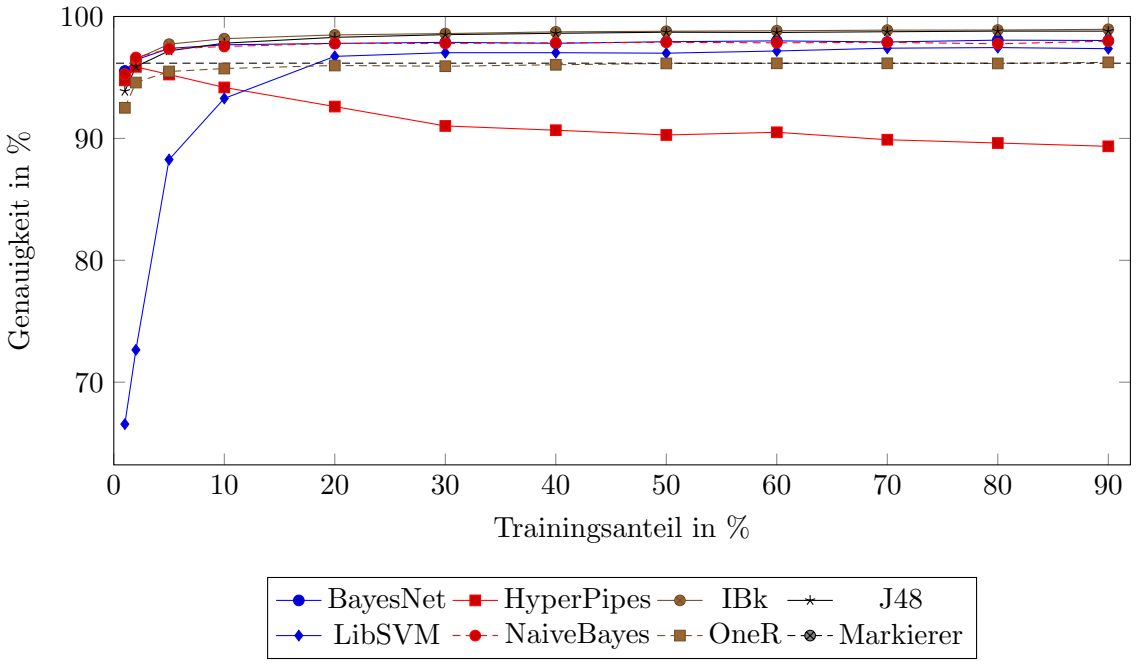

Abbildung 6.20.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte (Teil 2)

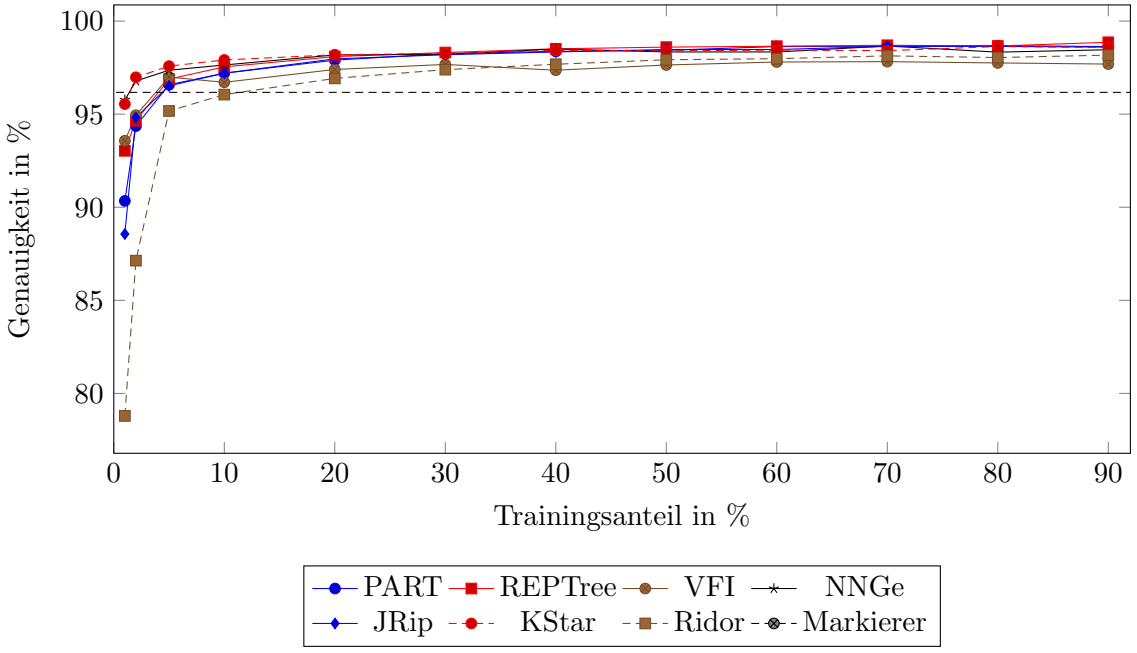

Abbildung 6.21.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 1)

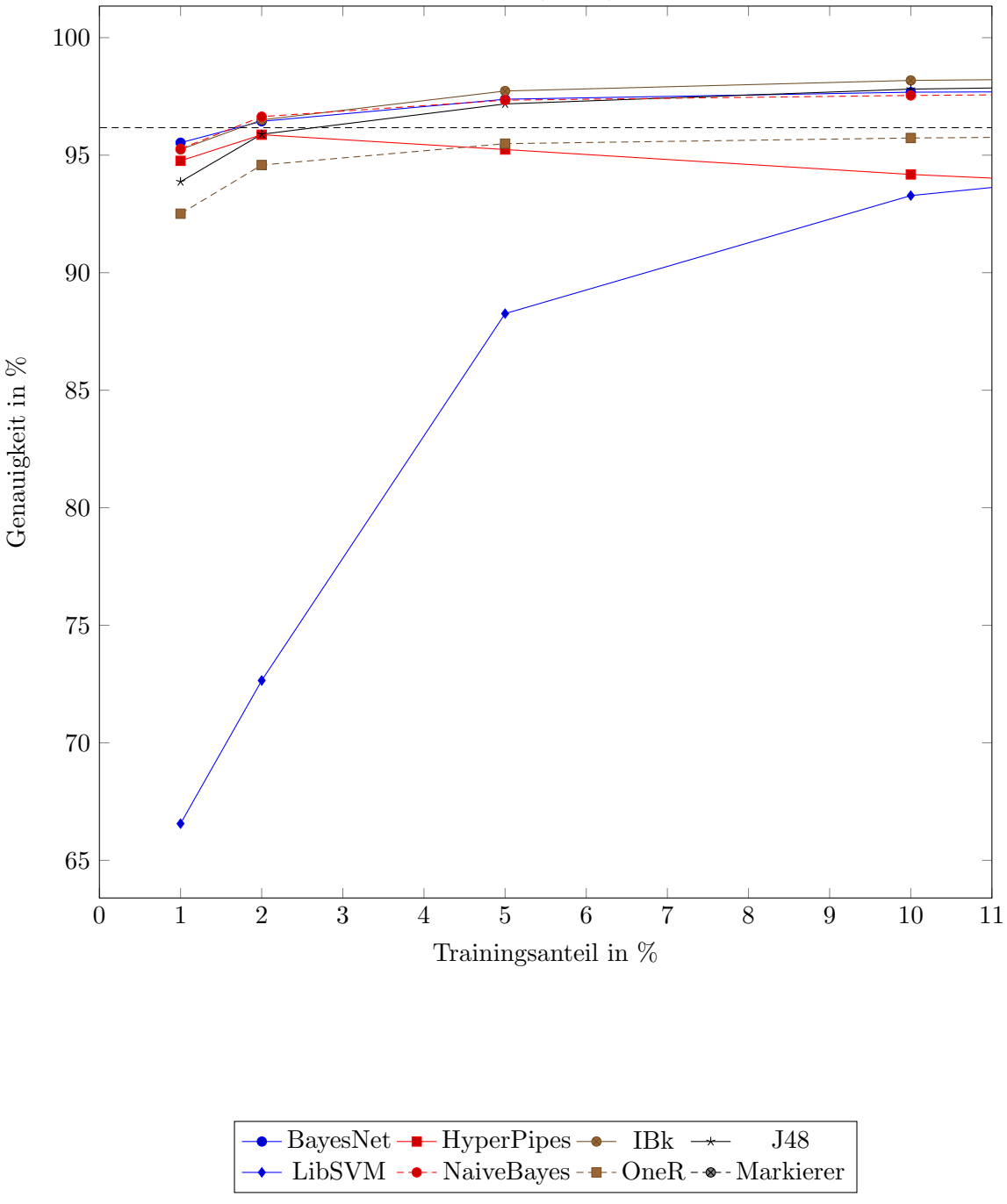

Abbildung 6.22.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 2)

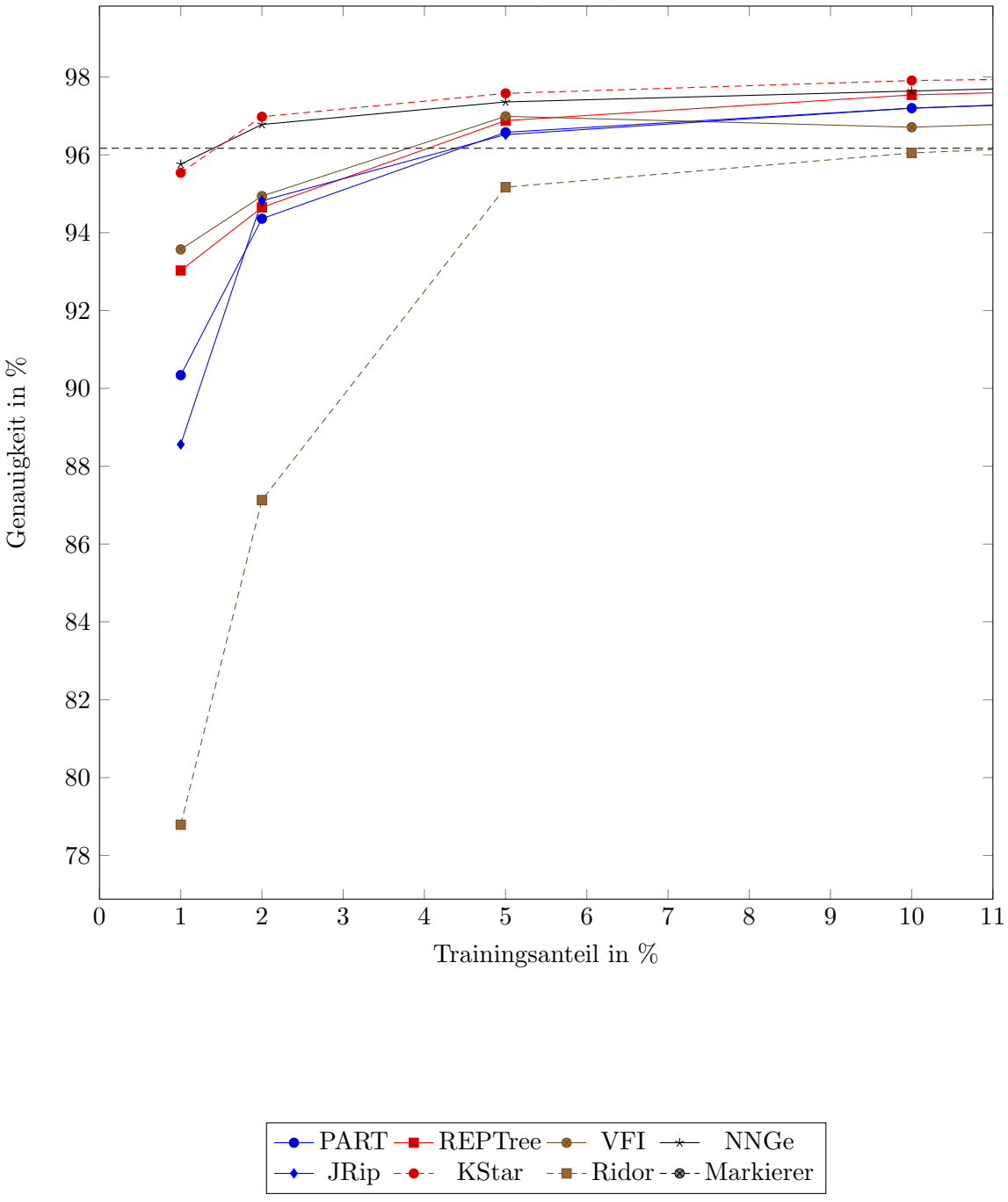

Abbildung 6.23.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte bei Trainingsanteil über 10% (Teil 1)

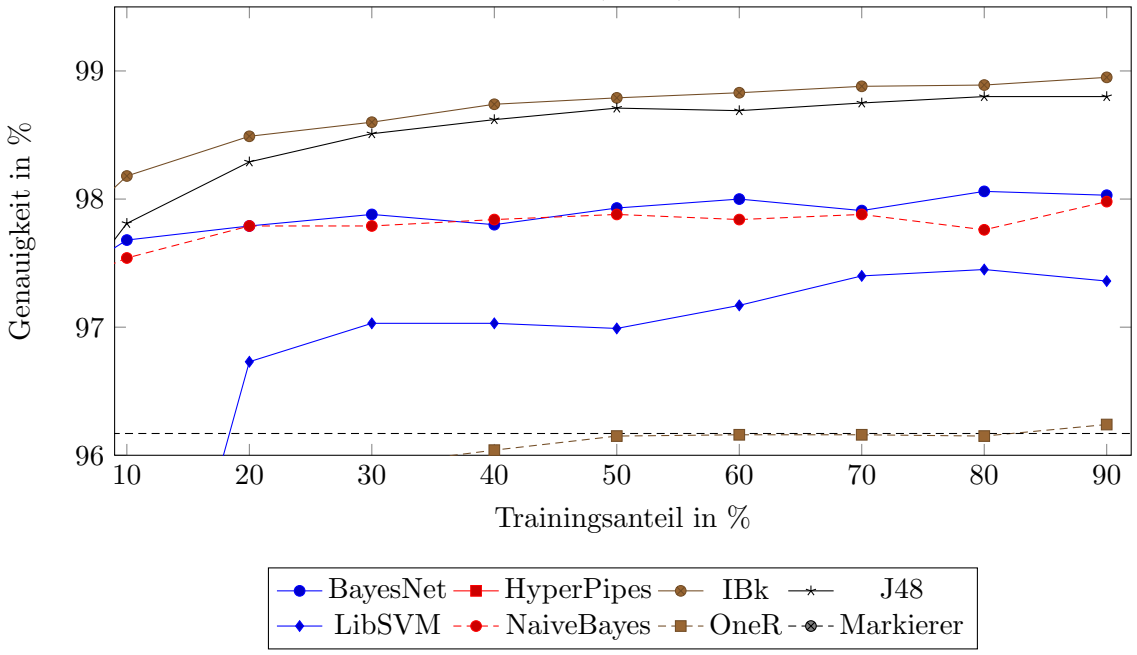

Abbildung 6.24.: Lernkurven der Klassifikationsverfahren auf NLCI-Korpus ohne Worte bei Trainingsanteil über 10% (Teil 2)

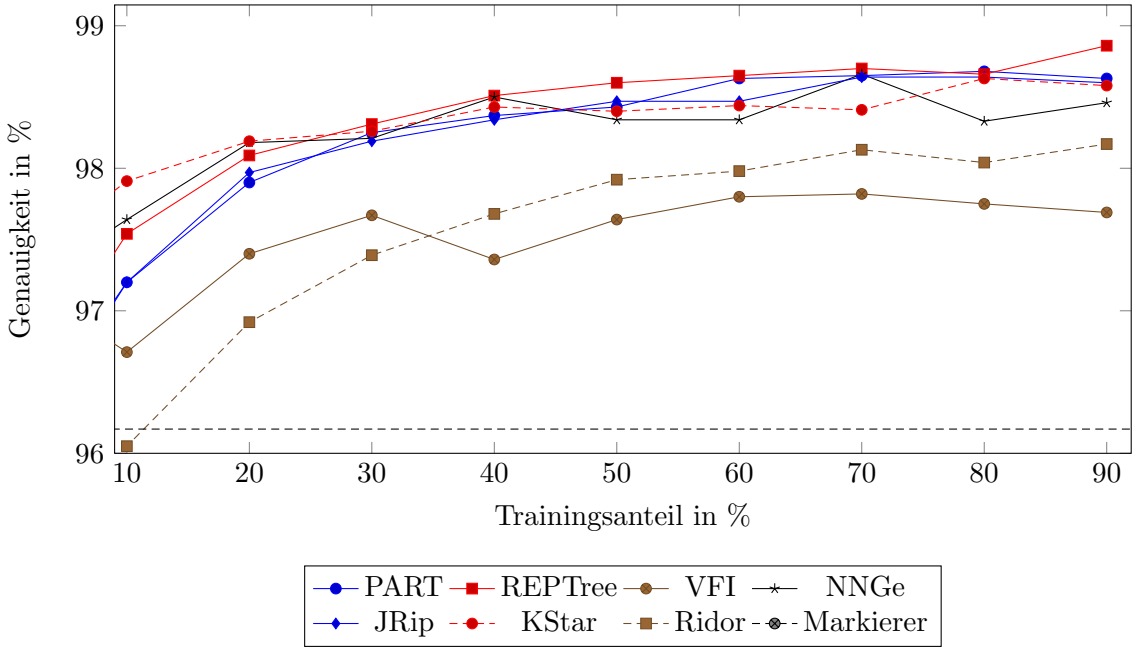

Abbildung 6.25.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten (Teil 1)

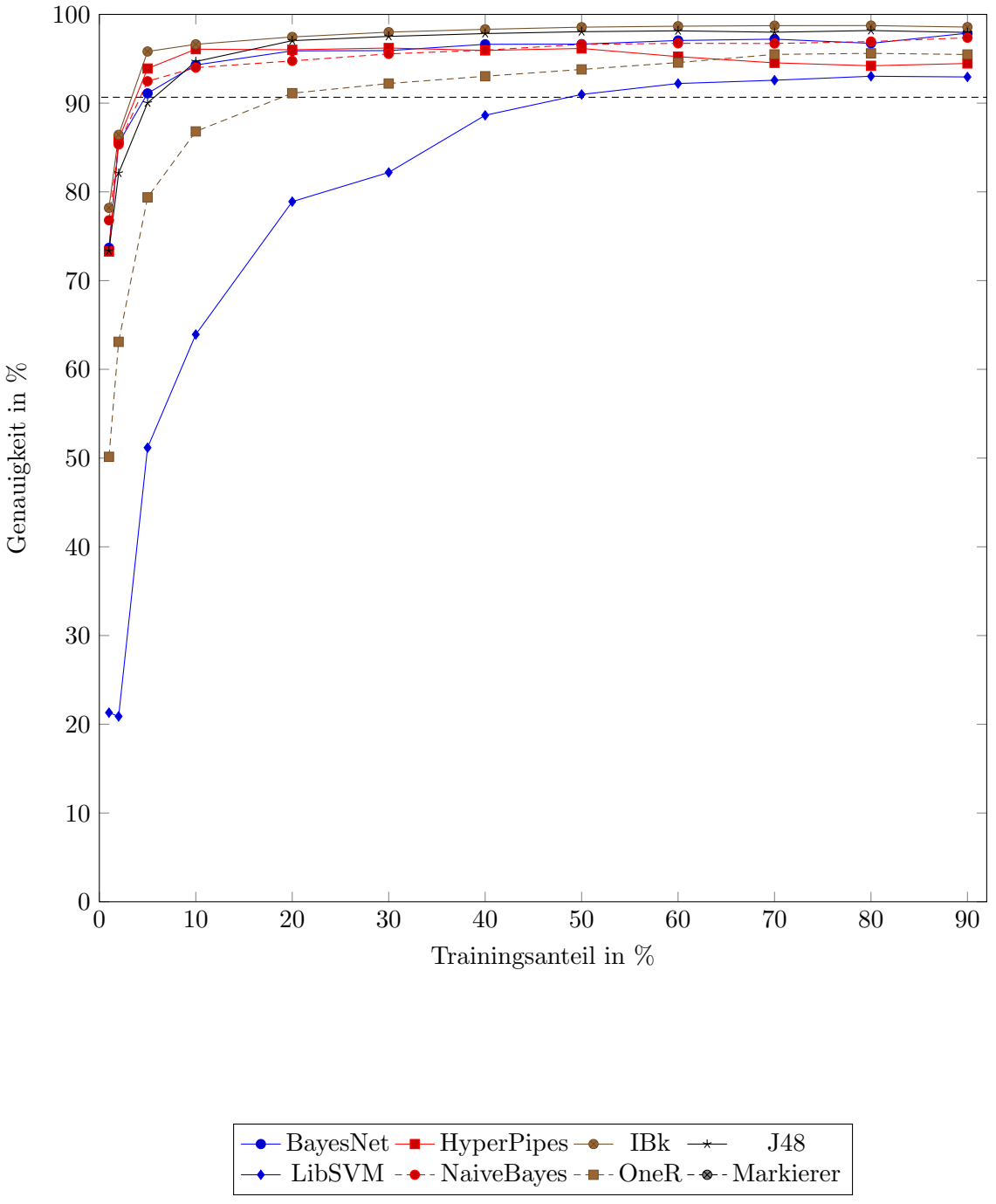

Abbildung 6.26.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten (Teil 2)

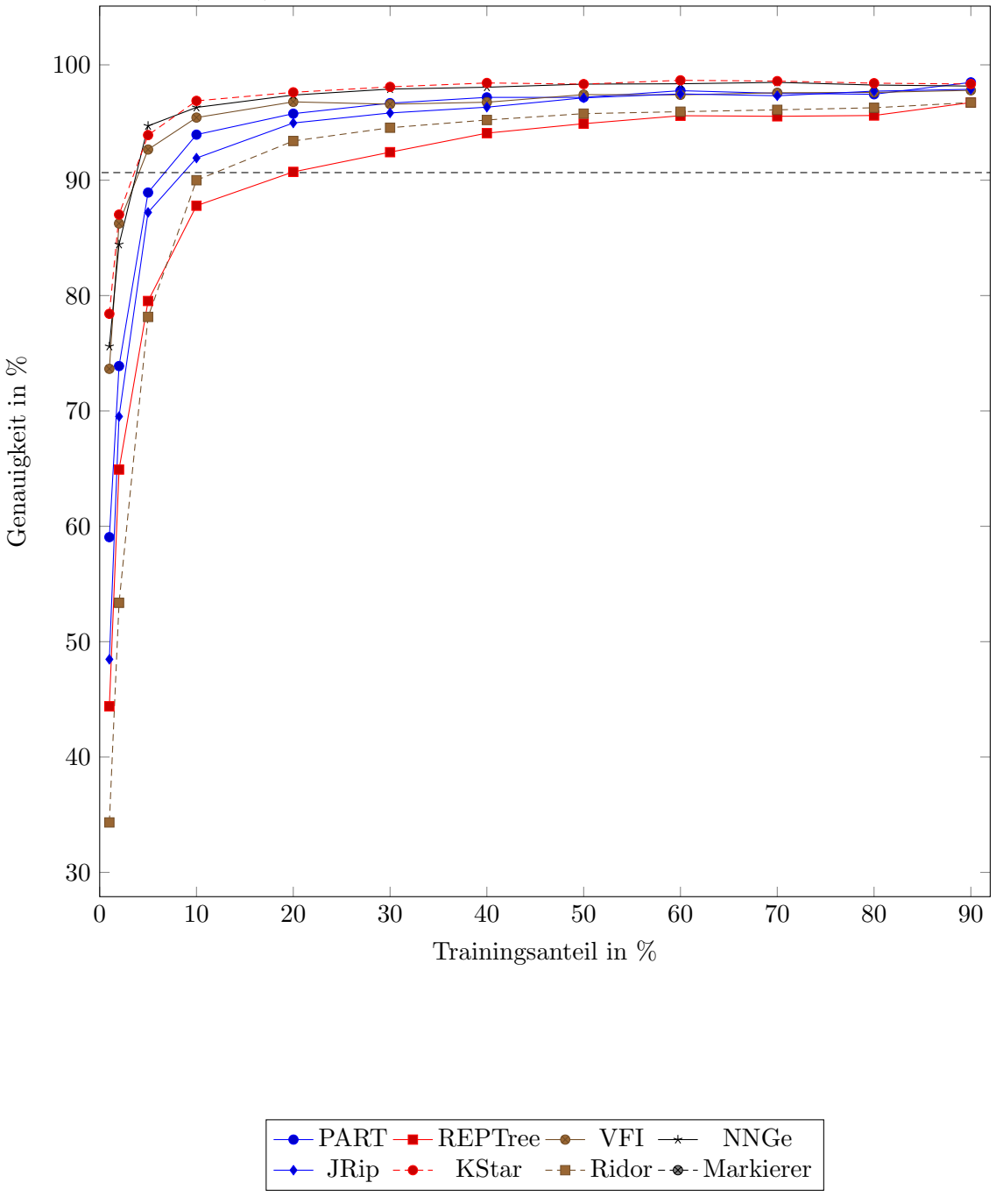

Abbildung 6.27.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 1)

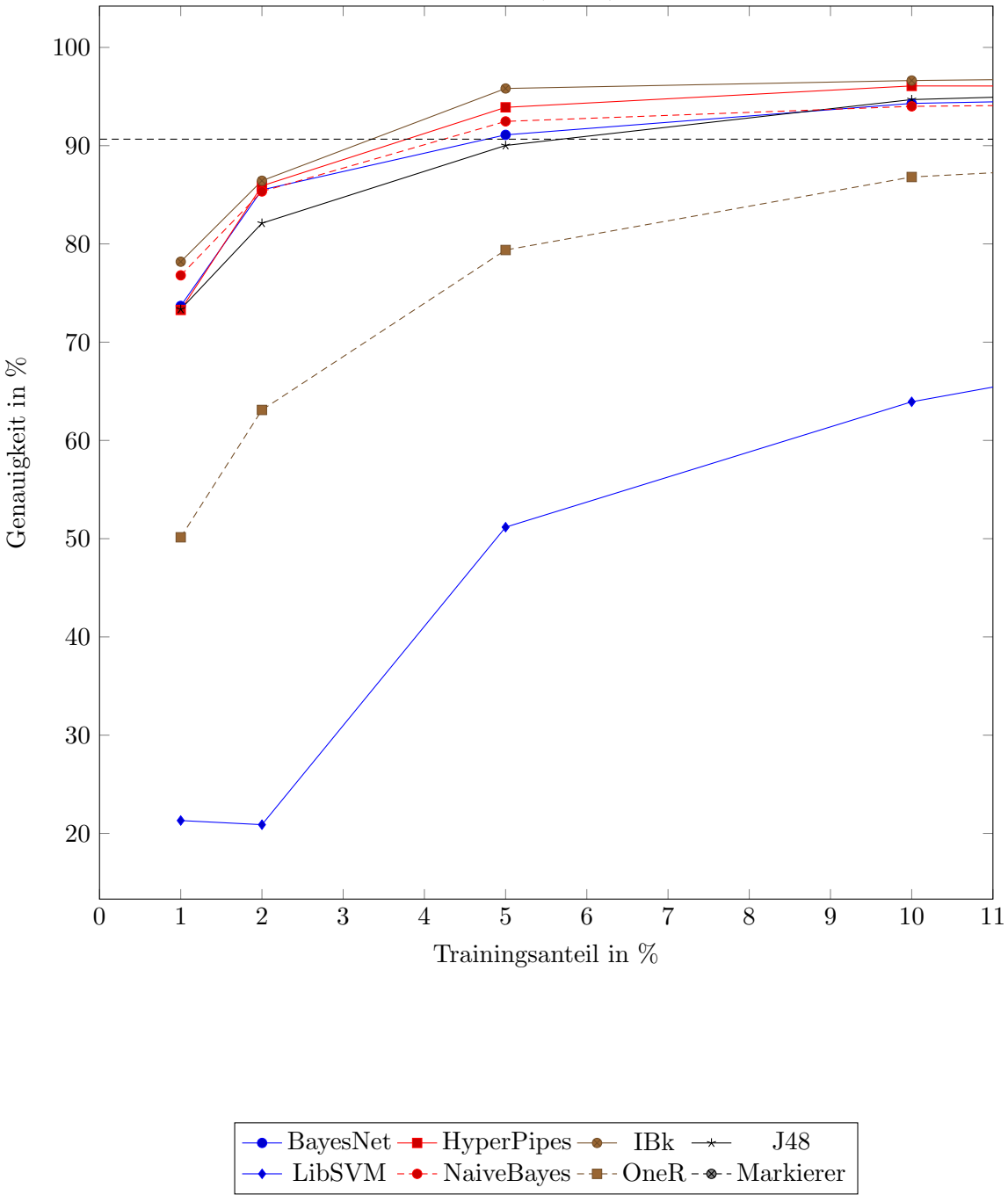

Abbildung 6.28.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten mit Trainingsanteil unter 10% (Teil 2)

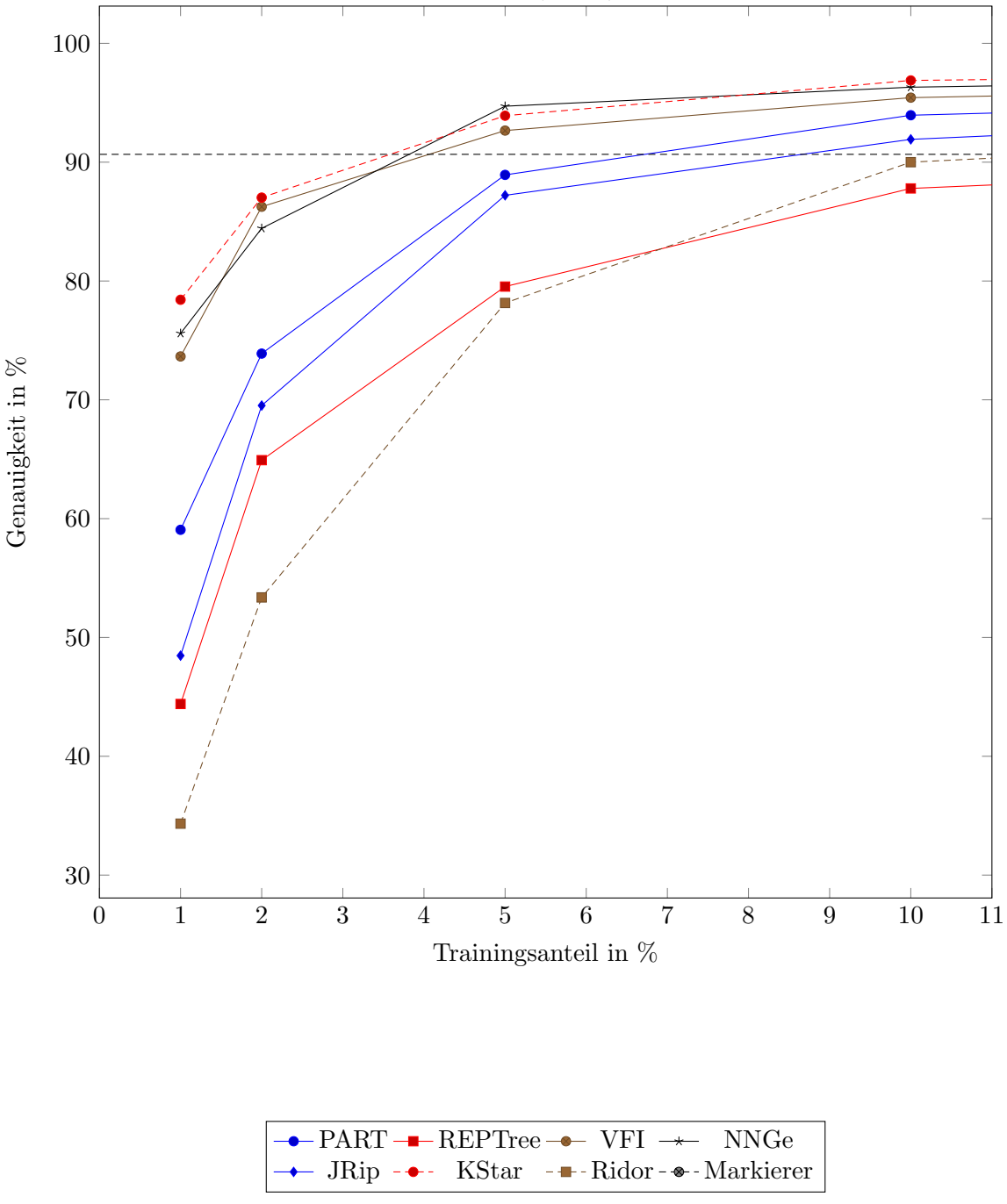

Abbildung 6.29.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten bei Trainingsanteil über 10% (Teil 1)

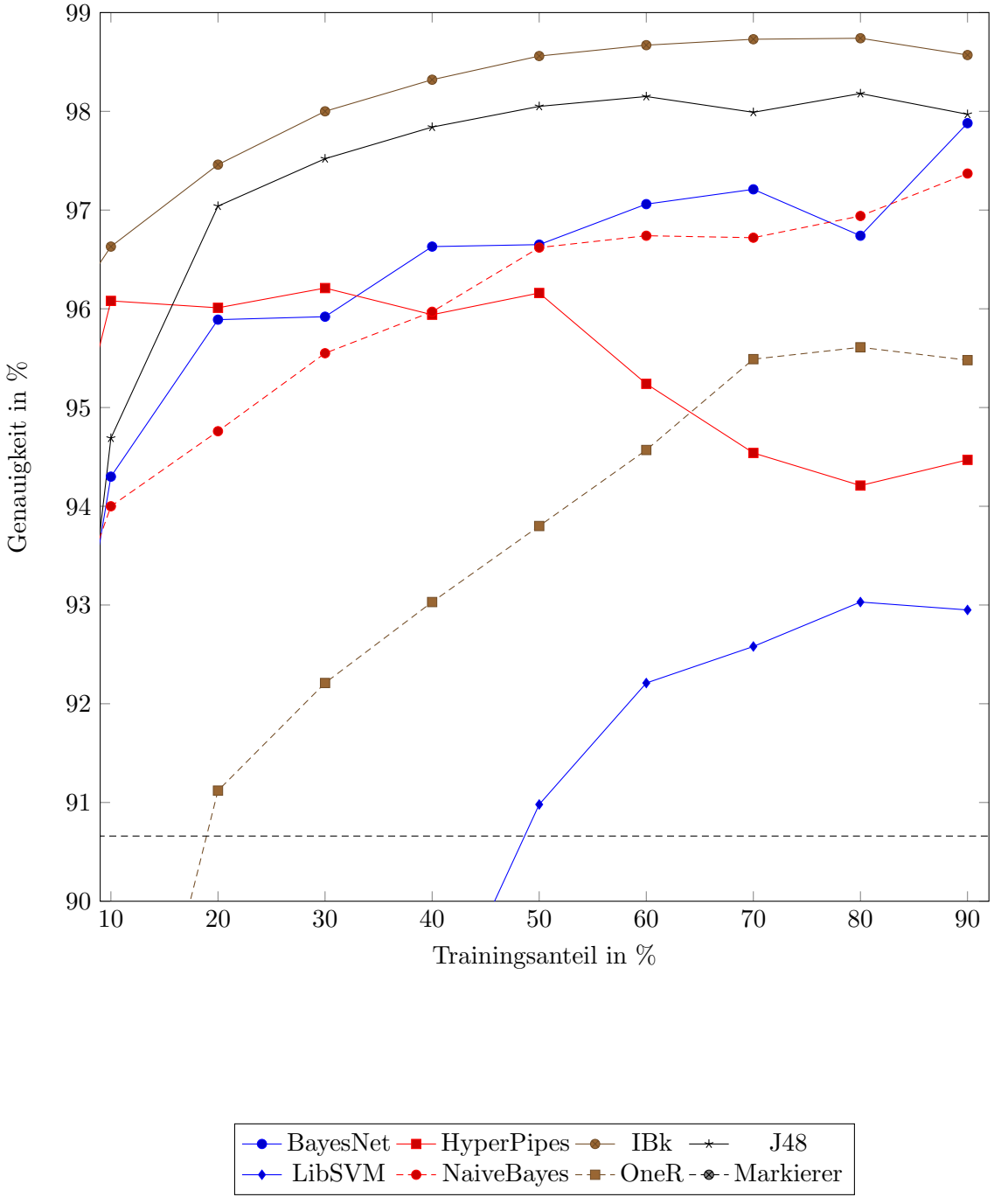

Abbildung 6.30.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus mit Worten bei Trainingsanteil über 10% (Teil 2)

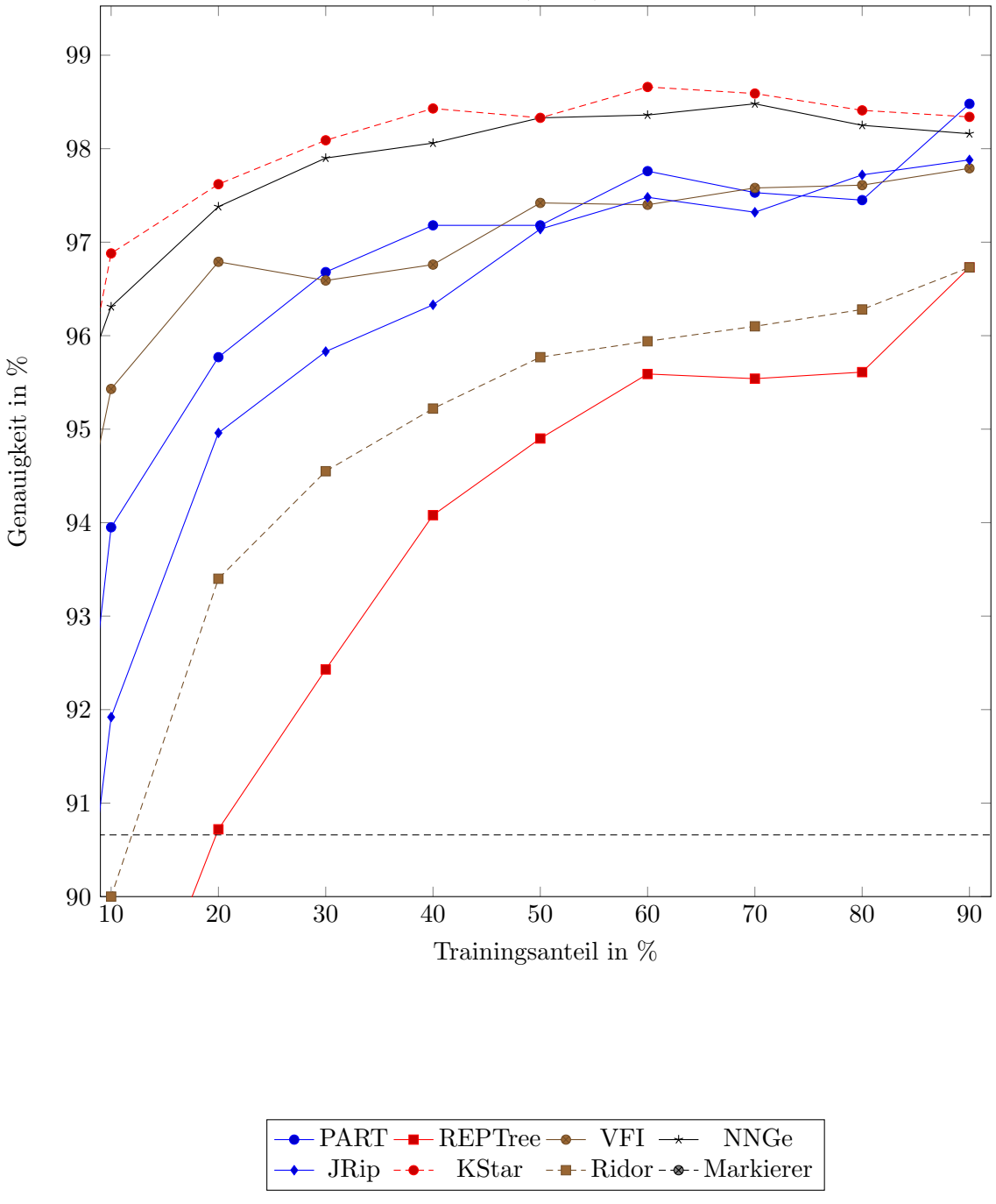

Abbildung 6.31.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte (Teil 1)

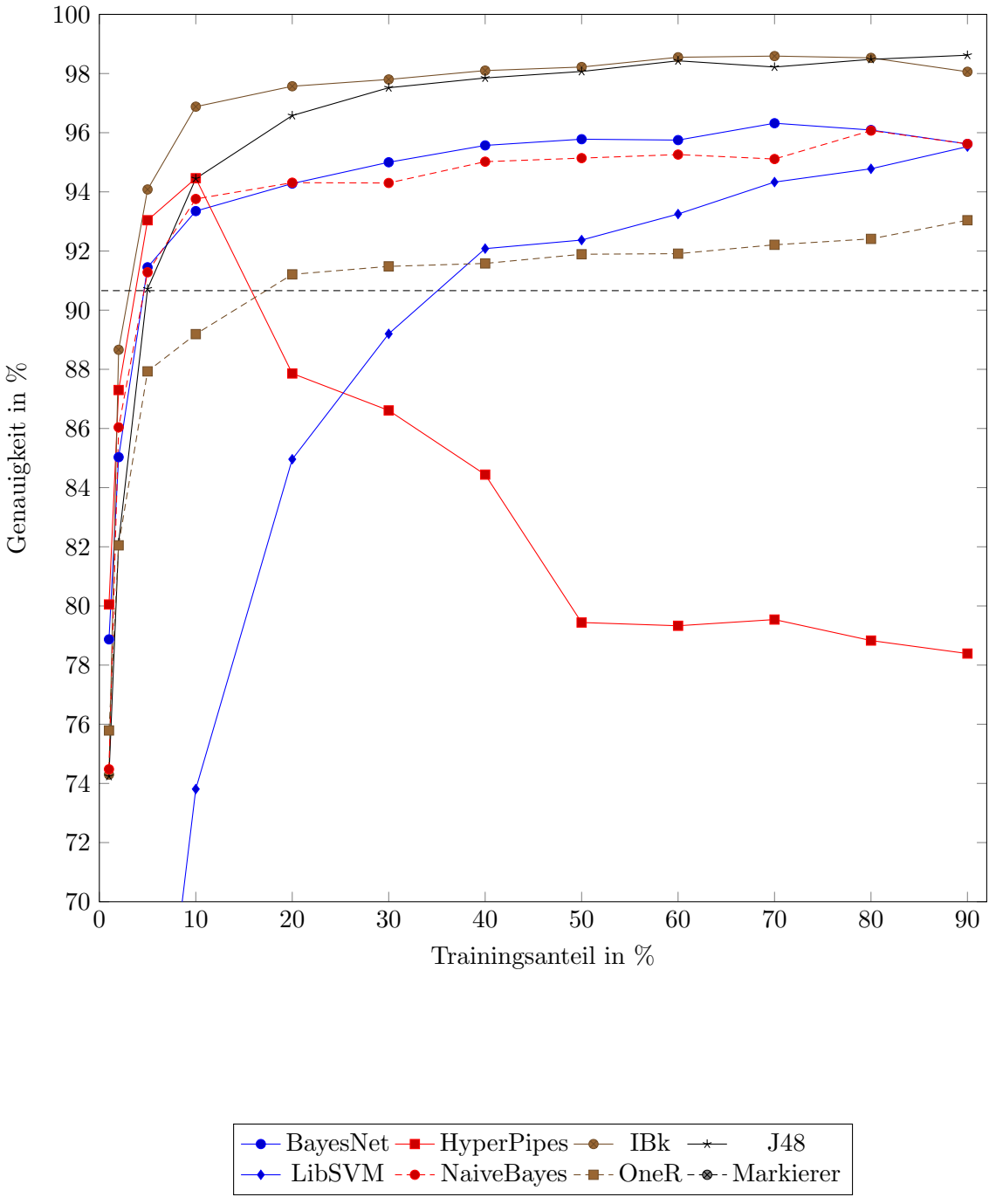

Abbildung 6.32.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte (Teil 2)

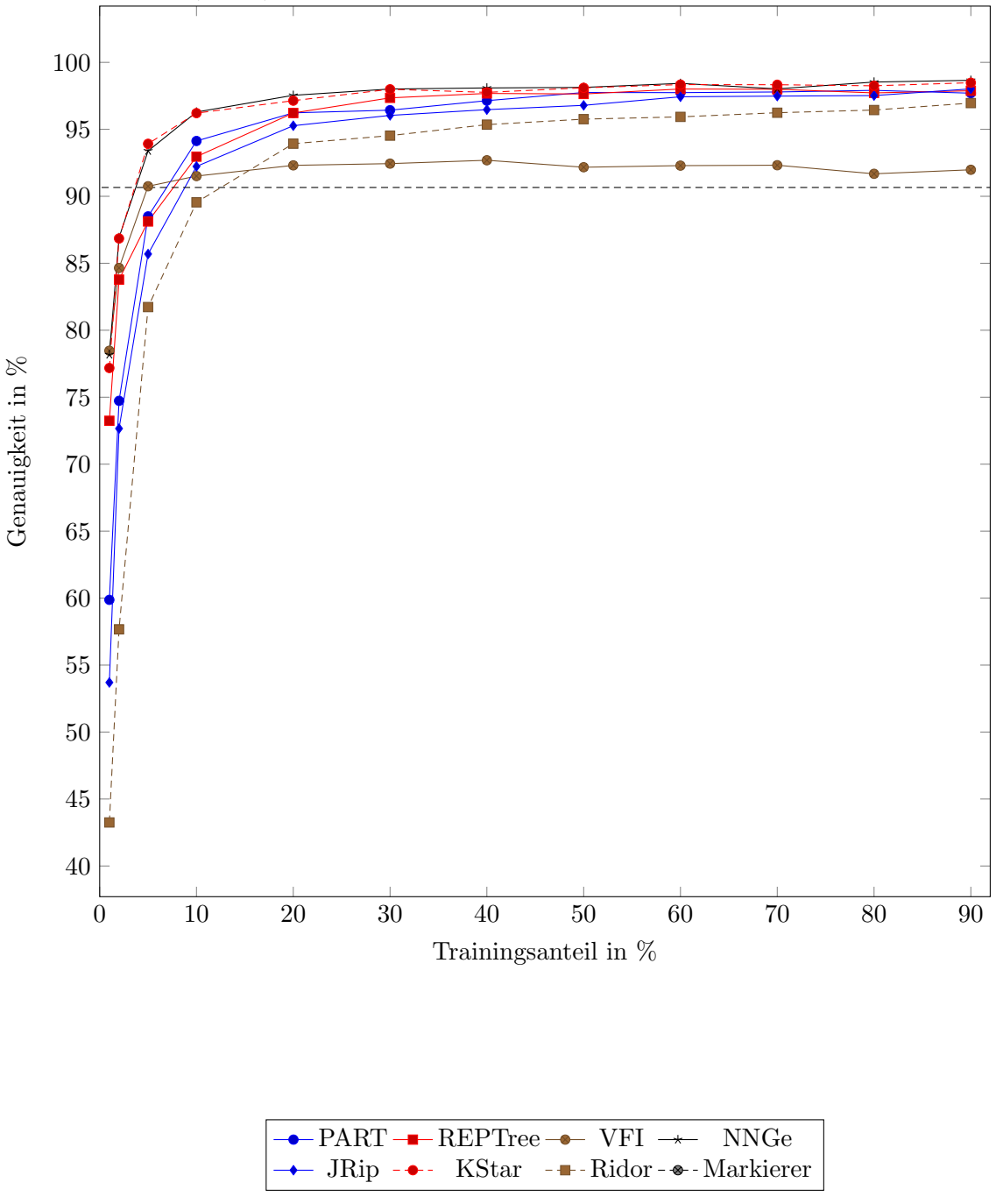

Abbildung 6.33.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 1)

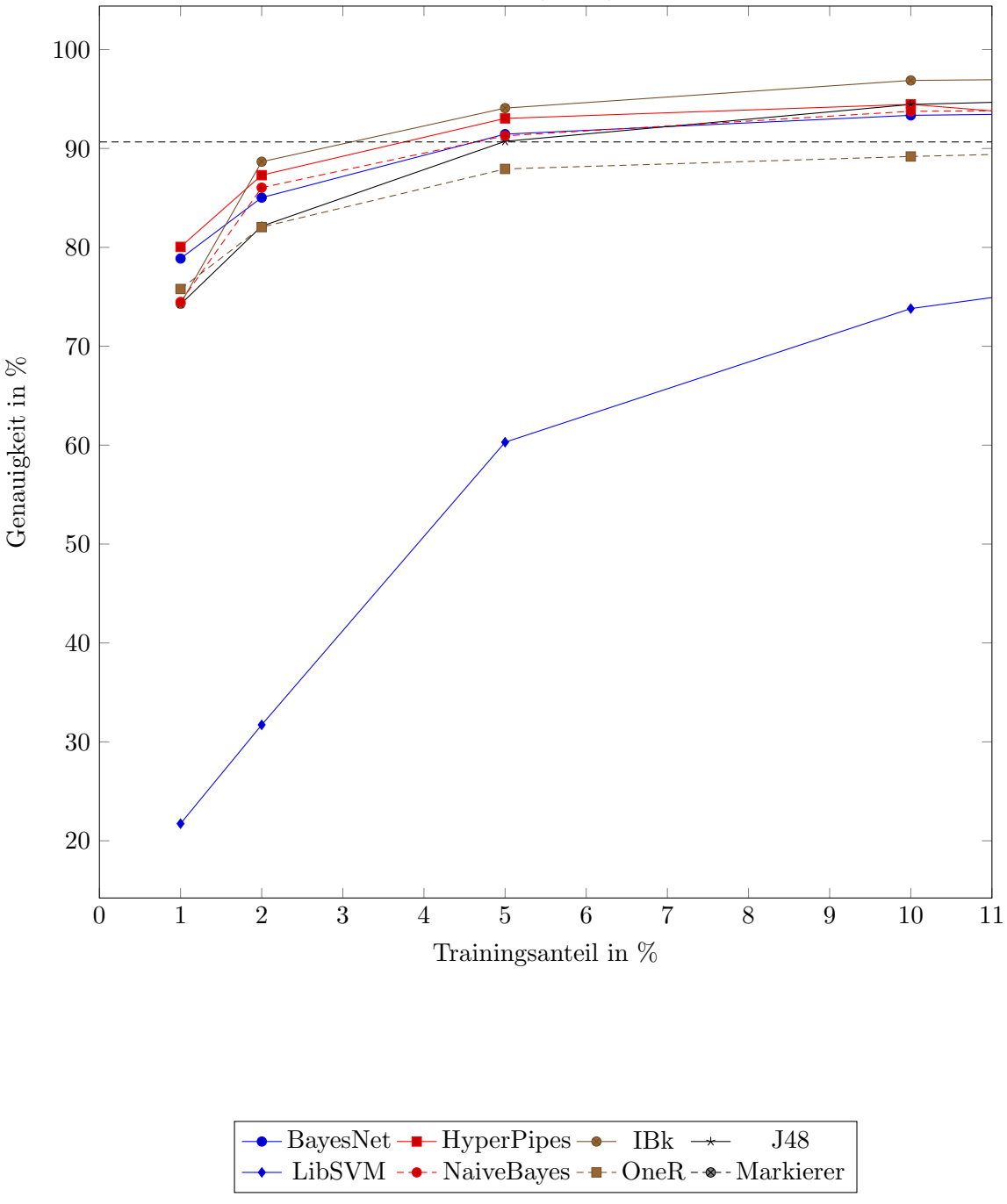

Abbildung 6.34.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte mit Trainingsanteil unter 10% (Teil 2)

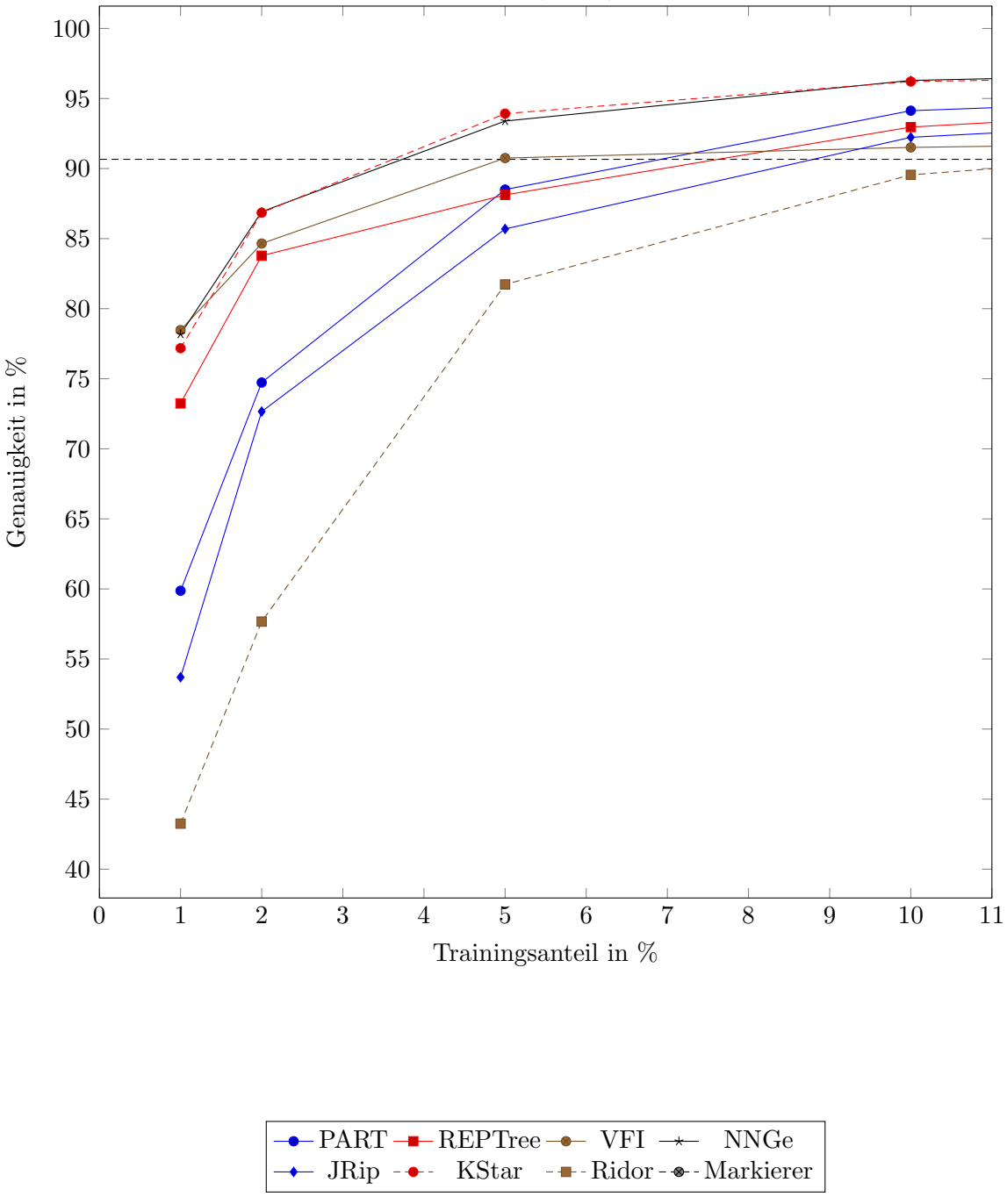

Abbildung 6.35.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte bei Trainingsanteil über 10% (Teil 1)

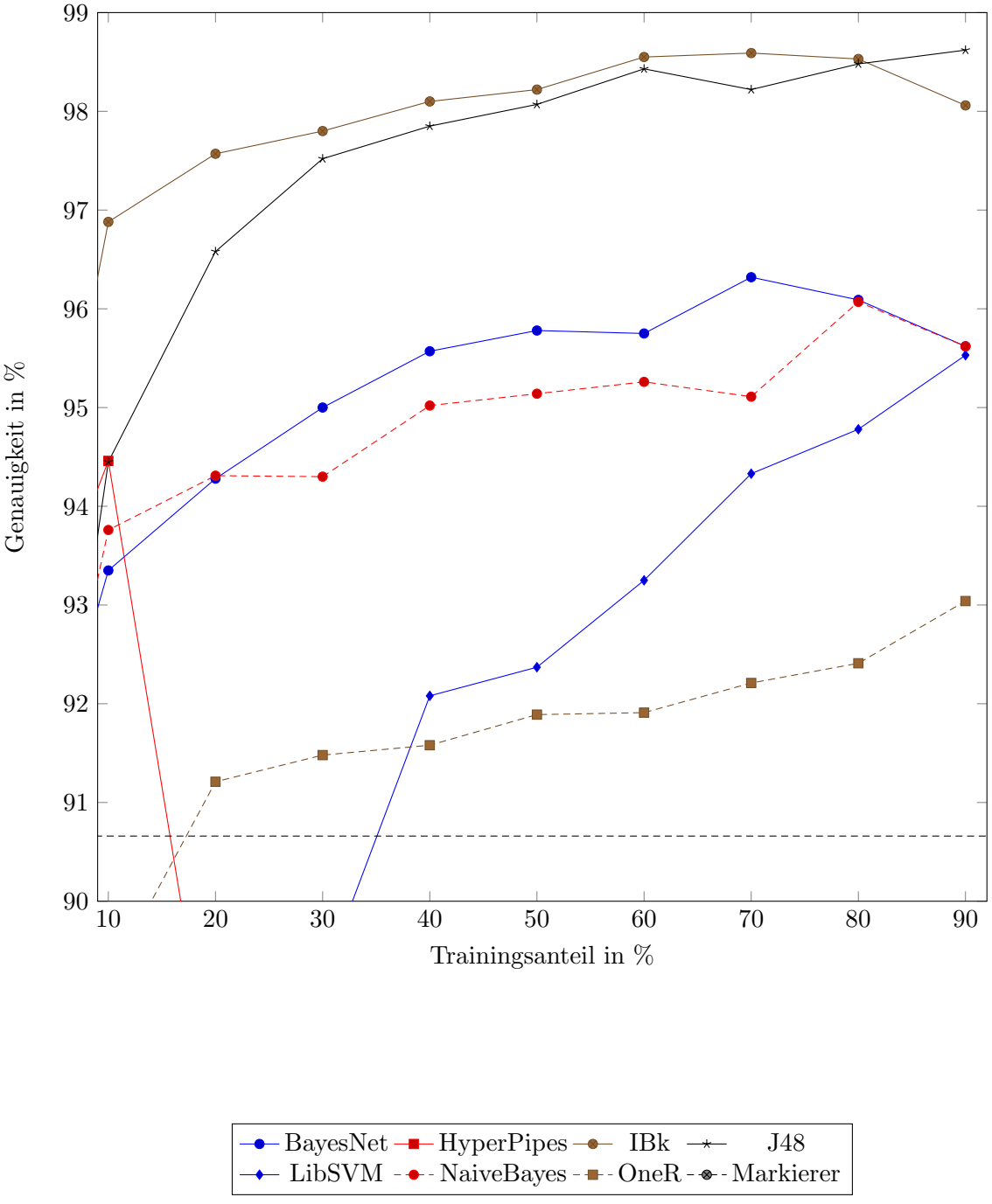

Abbildung 6.36.: Lernkurven der Klassifikationsverfahren auf PARSE-Korpus ohne Worte bei Trainingsanteil über 10% (Teil 2)

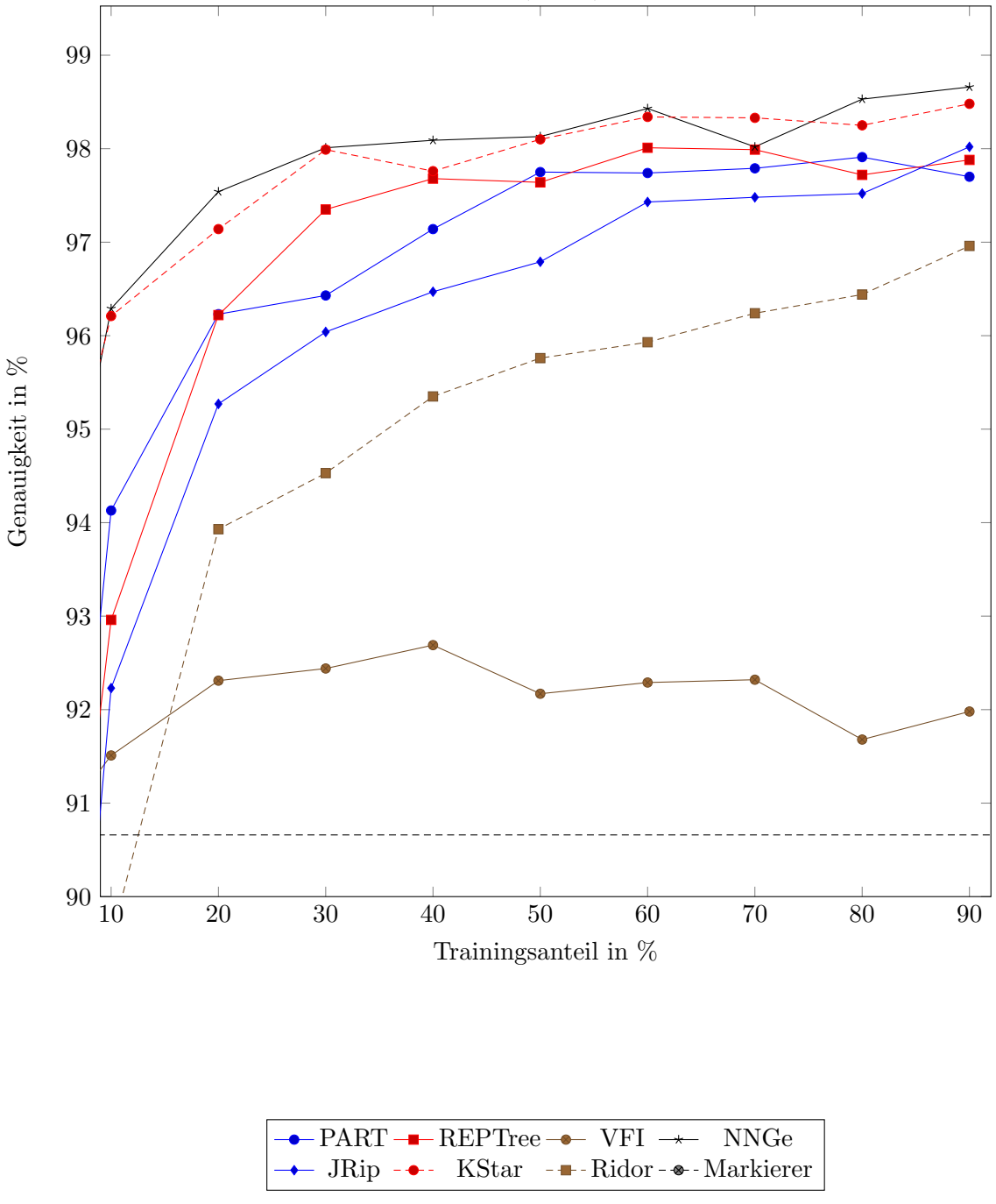

Tabelle 6.15.: Sättigungstabelle in Prozentpunkten für WSJ-Korpus mit Worten
Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(941)	(1882)	(4705)	(9409)	(18817)	(28226)	(37634)	(47043)	(56451)	(65860)	(75268)	(84677)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	97,80%	0,24%	0,11%	0,05%	0,06%	0,01%	-0,03%	0,02%	-0,03%	0,01%	-0,02%	0,03%
HyperPipes	93,78%	-1,97%	-9,46%	0,11%	-5,49%	-0,90%	-1,00%	-0,24%	-0,44%	-1,06%	-0,59%	-1,09%
IBk	97,69%	0,33%	0,22%	0,08%	0,07%	0,07%	0,03%	0,04%	0,00%	0,02%	0,01%	-0,02%
J48	97,77%	0,56%	0,19%	0,08%	0,06%	0,03%	0,02%	0,02%	0,00%	-0,02%	0,04%	0,10%
LibSVM	15,03%	-1,04%	34,25%	20,29%	18,64%	2,71%	1,05%	2,60%	1,75%	0,28%	0,27%	0,08%
NaiveBayes	97,58%	0,41%	0,15%	0,05%	0,02%	0,02%	0,01%	0,01%	-0,02%	0,06%	-0,04%	-0,05%
OneR	78,87%	9,67%	9,96%	0,02%	0,03%	0,00%	-0,02%	0,02%	0,00%	-0,01%	0,01%	-0,02%
PART	97,45%	0,40%	0,48%	0,15%	0,07%	0,08%	0,03%	-0,03%	0,03%	0,04%	0,06%	-0,06%
REPTree	65,76%	4,74%	3,85%	4,98%	4,55%	4,90%	4,40%	5,46%	0,04%	0,00%	0,01%	0,00%
VFI	94,75%	1,05%	-1,64%	-1,29%	1,12%	0,61%	0,22%	0,29%	0,24%	0,26%	0,08%	0,11%
NNge	93,29%	2,34%	0,27%	-1,53%	2,33%	0,22%	0,28%	0,45%	0,24%	-0,68%	0,44%	0,40%
JRip	96,70%	0,73%	0,71%	0,19%	0,17%	0,10%	0,02%	0,04%	0,01%	0,01%	0,01%	-0,07%
KStar	97,85%	0,24%	0,19%	0,07%	0,09%	0,02%	0,05%	0,01%	0,03%	-0,04%	0,00%	-0,01%
Ridor	93,43%	2,22%	1,20%	0,57%	0,41%	0,16%	0,17%	0,04%	0,06%	0,06%	0,02%	0,02%

Tabelle 6.16.: Sättigungstabelle in Prozentpunkten für WSJ-Korpus ohne Worte

Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(941)	(1882)	(4705)	(9409)	(18817)	(28226)	(37634)	(47043)	(56451)	(65860)	(75268)	(84677)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	97,81%	0,25%	0,09%	0,06%	0,04%	-0,02%	0,02%	0,00%	0,02%	-0,02%	-0,01%	-0,02%
HyperPipes	92,17%	-8,13%	-12,73%	-10,02%	-13,64%	-7,74%	-1,83%	-1,56%	-1,31%	-2,08%	-0,19%	-0,61%
IBk	97,88%	0,20%	0,19%	0,09%	0,05%	0,10%	-0,03%	0,06%	0,01%	0,03%	-0,03%	-0,02%
J48	98,04%	0,36%	0,10%	0,06%	0,06%	0,05%	0,02%	0,03%	-0,02%	0,01%	-0,01%	0,03%
LibSVM	86,30%	5,95%	4,03%	0,78%	0,74%	0,36%	0,07%	0,03%	0,03%	0,02%	0,05%	0,04%
NaiveBayes	97,83%	0,20%	0,13%	0,03%	0,03%	0,01%	0,00%	0,02%	-0,01%	0,00%	0,04%	-0,03%
OneR	98,02%	0,36%	0,12%	0,01%	0,03%	0,00%	0,01%	-0,03%	0,01%	0,02%	-0,03%	0,00%
PART	97,20%	0,66%	0,49%	0,09%	0,11%	0,05%	0,03%	0,01%	0,00%	0,02%	0,04%	-0,06%
REPTree	97,82%	0,44%	0,24%	0,04%	0,04%	0,04%	0,04%	-0,02%	0,02%	0,04%	-0,06%	0,06%
VFI	96,35%	-1,59%	-2,66%	-2,93%	-0,94%	-0,02%	-0,04%	0,04%	-0,01%	-0,11%	0,08%	0,10%
NNge	97,71%	0,26%	-0,07%	0,36%	-0,21%	0,12%	0,29%	0,00%	-0,09%	0,04%	-0,01%	0,04%
JRip	96,76%	0,75%	0,64%	0,14%	0,19%	0,02%	0,04%	0,00%	0,03%	0,02%	0,03%	-0,01%
KStar	97,84%	0,28%	0,15%	0,09%	0,06%	0,01%	0,03%	-0,01%	0,00%	0,01%	0,04%	-0,03%
Ridor	93,11%	2,46%	1,36%	0,61%	0,25%	0,25%	0,06%	0,06%	0,08%	0,03%	0,01%	0,00%

Tabelle 6.17.: Sättigungstabelle in Prozentpunkten für NLCI-Korpus mit Worten Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(182)	(363)	(907)	(1813)	(3625)	(5438)	(7250)	(9062)	(10875)	(12687)	(14500)	(16312)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	94,54%	2,33%	0,57%	0,27%	0,24%	0,21%	0,01%	0,06%	0,07%	0,00%	-0,07%	0,23%
HyperPipes	95,03%	1,53%	0,16%	-0,70%	-0,18%	-0,08%	-0,04%	-0,42%	-0,03%	-0,22%	-0,09%	-0,19%
IBk	95,77%	1,12%	0,98%	0,43%	0,33%	0,20%	0,06%	0,07%	0,03%	0,06%	0,11%	-0,05%
J48	93,65%	1,72%	1,69%	0,70%	0,50%	0,25%	0,01%	0,05%	0,04%	0,18%	-0,01%	-0,09%
LibSVM	37,44%	23,26%	10,36%	15,29%	5,19%	3,53%	1,02%	0,53%	0,21%	0,11%	-0,01%	-0,04%
NaiveBayes	94,53%	2,01%	0,78%	0,30%	0,21%	0,20%	0,07%	0,02%	0,14%	-0,07%	0,03%	-0,10%
OneR	68,00%	7,87%	9,80%	4,68%	2,83%	1,15%	0,56%	0,36%	0,31%	0,17%	0,31%	0,31%
PART	89,80%	4,05%	2,65%	0,79%	0,65%	0,26%	0,19%	-0,01%	0,24%	0,01%	0,10%	0,01%
REPTree	69,15%	10,05%	7,73%	4,22%	3,08%	1,24%	0,64%	0,31%	0,43%	0,24%	0,23%	0,07%
VFI	94,63%	1,72%	0,94%	0,64%	0,24%	0,18%	0,03%	0,14%	-0,05%	0,17%	-0,05%	0,00%
NNge	93,84%	2,65%	0,49%	0,64%	0,51%	0,34%	0,05%	0,11%	0,18%	-0,10%	0,10%	0,16%
JRip	86,36%	7,03%	2,96%	0,76%	0,79%	0,39%	0,16%	0,08%	0,12%	0,03%	-0,04%	0,00%
KStar	95,06%	1,55%	1,03%	0,61%	0,36%	0,18%	0,05%	0,07%	0,08%	0,04%	0,01%	0,01%
Ridor	80,41%	8,08%	6,17%	1,17%	1,05%	0,53%	0,32%	0,27%	0,04%	0,14%	0,14%	0,08%

Tabelle 6.18.: Sättigungstabelle in Prozentpunkten für NLCI-Korpus ohne Worte
Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(182)	(363)	(907)	(1813)	(3625)	(5438)	(7250)	(9062)	(10875)	(12687)	(14500)	(16312)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	95,53%	0,91%	0,94%	0,30%	0,11%	0,09%	-0,08%	0,13%	0,07%	-0,09%	0,15%	-0,03%
HyperPipes	94,76%	1,11%	-0,63%	-1,06%	-1,57%	-1,59%	-0,35%	-0,39%	0,22%	-0,61%	-0,27%	-0,27%
IBk	95,24%	1,26%	1,23%	0,45%	0,31%	0,11%	0,14%	0,05%	0,04%	0,05%	0,01%	0,06%
J48	93,87%	2,02%	1,29%	0,63%	0,48%	0,22%	0,11%	0,09%	-0,02%	0,06%	0,05%	0,00%
LibSVM	66,56%	6,09%	15,61%	5,02%	3,45%	0,30%	0,00%	-0,04%	0,18%	0,23%	0,05%	-0,09%
NaiveBayes	95,27%	1,37%	0,70%	0,20%	0,25%	0,00%	0,05%	0,04%	-0,04%	0,04%	-0,12%	0,22%
OneR	92,51%	2,07%	0,91%	0,24%	0,25%	-0,06%	0,12%	0,11%	0,01%	0,00%	-0,01%	0,09%
PART	90,34%	4,02%	2,22%	0,62%	0,70%	0,35%	0,12%	0,06%	0,20%	0,02%	0,03%	-0,05%
REPTree	93,03%	1,62%	2,23%	0,66%	0,55%	0,22%	0,20%	0,09%	0,05%	0,05%	-0,04%	0,20%
VFI	93,57%	1,37%	2,05%	-0,28%	0,69%	0,27%	-0,31%	0,28%	0,16%	0,02%	-0,07%	-0,06%
NNge	95,76%	1,02%	0,58%	0,28%	0,54%	0,03%	0,29%	-0,16%	0,00%	0,32%	-0,33%	0,13%
JRip	88,56%	6,26%	1,70%	0,68%	0,77%	0,22%	0,15%	0,13%	0,00%	0,17%	0,00%	-0,04%
KStar	95,54%	1,44%	0,60%	0,33%	0,28%	0,07%	0,17%	-0,03%	0,04%	-0,03%	0,22%	-0,05%
Ridor	78,79%	8,34%	8,04%	0,88%	0,87%	0,47%	0,29%	0,24%	0,06%	0,15%	-0,09%	0,13%

Tabelle 6.19.: Sättigungstabelle in Prozentpunkten für PARSE-Korpus mit Worten Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(22)	(44)	(109)	(218)	(435)	(652)	(870)	(1087)	(1304)	(1522)	(1739)	(1956)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	73,69%	11,81%	5,60%	3,20%	1,59%	0,03%	0,71%	0,02%	0,41%	0,15%	-0,47%	1,14%
HyperPipes	73,28%	12,63%	7,99%	2,18%	-0,07%	0,20%	-0,27%	0,22%	-0,92%	-0,70%	-0,33%	0,26%
IBk	78,19%	8,24%	9,39%	0,81%	0,83%	0,54%	0,32%	0,24%	0,11%	0,06%	0,01%	-0,17%
J48	73,35%	8,76%	7,91%	4,67%	2,35%	0,48%	0,32%	0,21%	0,10%	-0,16%	0,19%	-0,21%
LibSVM	21,31%	-0,42%	30,28%	12,76%	14,97%	3,29%	6,44%	2,35%	1,23%	0,37%	0,45%	-0,08%
NaiveBayes	76,80%	8,54%	7,13%	1,53%	0,76%	0,79%	0,42%	0,65%	0,12%	-0,02%	0,22%	0,43%
OneR	50,14%	12,96%	16,28%	7,43%	4,31%	1,09%	0,82%	0,77%	0,77%	0,92%	0,12%	-0,13%
PART	59,06%	14,83%	15,04%	5,02%	1,82%	0,91%	0,50%	0,00%	0,58%	-0,23%	-0,08%	1,03%
REPTree	44,40%	20,52%	14,61%	8,26%	2,93%	1,71%	1,65%	0,82%	0,69%	-0,05%	0,07%	1,12%
VFI	73,65%	12,60%	6,41%	2,77%	1,36%	-0,20%	0,17%	0,66%	-0,02%	0,18%	0,03%	0,18%
NNge	75,59%	8,84%	10,28%	1,60%	1,07%	0,52%	0,16%	0,27%	0,03%	0,12%	-0,23%	-0,09%
JRip	48,47%	21,05%	17,69%	4,71%	3,04%	0,87%	0,50%	0,81%	0,34%	-0,16%	0,40%	0,16%
KStar	78,42%	8,60%	6,89%	2,97%	0,74%	0,47%	0,34%	-0,10%	0,33%	-0,07%	-0,18%	-0,07%
Ridor	34,33%	19,04%	24,78%	11,85%	3,40%	1,15%	0,67%	0,55%	0,17%	0,16%	0,18%	0,45%

Tabelle 6.20.: Sättigungstabelle in Prozentpunkten für PARSE-Korpus ohne Worte

Wachstum abhängig vom Trainingsanteil (Wortgröße des Trainingskorpus)

	(22)	(44)	(109)	(218)	(435)	(652)	(870)	(1087)	(1304)	(1522)	(1739)	(1956)
Verfahren	1,00%	2,00%	5,00%	10,00%	20,00%	30,00%	40,00%	50,00%	60,00%	70,00%	80,00%	90,00%
BayesNet	78,87%	6,16%	6,42%	1,90%	0,93%	0,72%	0,57%	0,21%	-0,03%	0,57%	-0,23%	-0,47%
HyperPipes	80,05%	7,25%	5,74%	1,42%	-6,60%	-1,25%	-2,17%	-5,00%	-0,11%	0,21%	-0,71%	-0,44%
IBk	74,30%	14,36%	5,42%	2,80%	0,69%	0,23%	0,30%	0,12%	0,33%	0,04%	-0,06%	-0,47%
J48	74,24%	7,90%	8,58%	3,73%	2,13%	0,94%	0,33%	0,22%	0,36%	-0,21%	0,26%	0,14%
LibSVM	21,73%	9,99%	28,58%	13,51%	11,15%	4,24%	2,88%	0,29%	0,88%	1,08%	0,45%	0,75%
NaiveBayes	74,48%	11,56%	5,24%	2,48%	0,55%	-0,01%	0,72%	0,12%	0,12%	-0,15%	0,96%	-0,45%
OneR	75,79%	6,26%	5,88%	1,26%	2,02%	0,27%	0,10%	0,31%	0,02%	0,30%	0,20%	0,63%
PART	59,87%	14,86%	13,77%	5,63%	2,10%	0,20%	0,71%	0,61%	-0,01%	0,05%	0,12%	-0,21%
REPTree	73,24%	10,54%	4,34%	4,84%	3,26%	1,13%	0,33%	-0,04%	0,37%	-0,02%	-0,27%	0,16%
VFI	78,47%	6,18%	6,10%	0,76%	0,80%	0,13%	0,25%	-0,52%	0,12%	0,03%	-0,64%	0,30%
NNge	78,18%	8,74%	6,47%	2,90%	1,25%	0,47%	0,08%	0,04%	0,30%	-0,41%	0,51%	0,13%
JRip	53,70%	18,96%	13,03%	6,54%	3,04%	0,77%	0,43%	0,32%	0,64%	0,05%	0,04%	0,50%
KStar	77,18%	9,67%	7,07%	2,29%	0,93%	0,85%	-0,23%	0,34%	0,24%	-0,01%	-0,08%	0,23%
Ridor	43,25%	14,42%	24,06%	7,82%	4,38%	0,60%	0,82%	0,41%	0,17%	0,31%	0,20%	0,52%

7. Zusammenfassung und Ausblick

Diese Arbeit beschäftigte sich mit der Optimierung von Wortartmarkierern. Die werden in der Computerlinguistik oft im ersten Verarbeitungsschritt für viele Anwendungen eingesetzt. Zur Vermeidung von Folgefehler, die Projekte durchgehend beeinträchtigen, sind möglichst optimale Wortartmarkierer nötig. Weil nicht immer ein optimaler Markierer zur Verfügung steht, wurde eine Alternativmöglichkeit untersucht, die Ergebnisse der Wortartmarkierung zu verbessern.

In dieser Arbeit wurde gezeigt, dass es möglich ist, mehrere Wortartmarkierer durch Klassifikationsverfahren zu kombinieren. Dafür wurden neun Wortartmarkierer und die drei Korpora WSJ, NLCI und PARSE verwendet. Zwei Wortartmarkierer nutzten hierbei je zwei verschiedene Modelle. Dadurch standen jedem Klassifikationsverfahren elf Markierungen und das Wort selbst als Attribut zur Verfügung. Die Verfahren wurden mit dem jeweils besten verwendeten Markierer und der einfachen Mehrheitswahl verglichen. Dabei konnten für jeden Korpus Verfahren gefunden werden, die besser abschneiden als beide Vergleichsverfahren. Auf dem WSJ-Korpus, der auf Texten einer Wirtschaftszeitung basiert, übertraf das beste Klassifikationsverfahren J48 mit 98,85% den Vergleichsmarkierer um 0,31 Prozentpunkte. Eine Verbesserung von 2,94 Prozentpunkten zum Vergleichsmarkierer fand auf dem NLCI-Korpus durch das Verfahren IBk mit 99,11% statt. Selbiges erreichte auf dem PARSE-Korpus mit 98,66% und 8,00 Prozentpunkten Vorsprung auch die größte Verbesserung.

Es wurde auch analysiert, welche Auswirkung das Hinzufügen der Worte als Attribut auf die Klassifikationsverfahren hat. Dabei konnte festgestellt werden, dass die Verfahren von den Worten profitieren können. So ist der PARSE-Korpus der einzige Korpus, dessen bestes Verfahren die Worte nicht nutzt. Weiterhin wurde untersucht wie sich die Trainingsgröße auswirkt. Generell zeigten sich mit größeren Trainingsmengen Verbesserungen, doch existieren Verfahren gegenteiligen Verhaltens. Außerdem konnte bei allen Verfahren eine Sättigung beobachtet werden.

Um dieses Ergebnis zu erreichen wurden zwei Programme geschrieben. Eins koordiniert Wortartmarkierer und sammelt die Ergebnisse der Markierung. Im zweiten werden diese Ergebnisse dazu benutzt, Modelle für Klassifikationsverfahren zu trainieren, zu evaluieren und Texte mit diesen Modellen zu klassifizieren. Beide Programme sind modular aufgebaut und lassen sich leicht erweitern.

Auf dieser Arbeit aufbauend bieten sich Ideen für weiterführende wissenschaftliche Arbeiten an. Einige Ideen sollen hier vorgestellt werden.

Aus Zeitgründen konnten nicht alle öffentlich verfügbaren Wortartmarkierer vermessen und zur Kombination verwendet werden. Ein offensichtlicher weiterer Forschungsschritt wäre folglich, die noch nicht betrachteten Markierer aus Abschnitt 4.3 einzubinden. Weitere Wortartmarkierer sollten das Ergebnis noch weiter verbessern. Interessant zu wissen wäre außerdem, wie sich die Anzahl der verwendeten Wortartmarkierer auf das Ergebnis der Kombination auswirkt und wie viele Wortartmarkierer mindestens gebraucht werden, um eine Ergebnisverbesserung zu erhalten.

Auch die Auswahl der Markierer ist von Interesse. Plausibel ist, dass verschiedene Markierer das Ergebnis verschieden stark verbessern. Es wäre daher nützlich, eine Heuristik zu finden, die erlaubt, nur die besten Wortartmarkierer zu verwenden.

Abseits der Wortartmarkierer ließe sich das Programm leicht durch neue Ausgabemodule erweitern. Eins dieser Module könnte Worte nicht nur isoliert betrachten, sondern auch den Wortkontext. Dadurch würde ermöglicht, weitere Problemfälle richtig zu klassifizieren. In dieser Arbeit stieß das Verfahren mit 16GB Arbeitsspeicher beim Klassifizieren bereits an seine Grenzen. Da das Hinzunehmen des Wortkontextes den benötigten Speicher vervielfachen würde ist ungewiss, ob heutige Technik dazu in der Lage ist.

Eine weitere Ausgabe könnte so erzeugt werden, dass Worte in ihre Merkmale aufgespalten werden. Als Attribute verwendet könnten sie zu einem robusteren und genaueren Verfahren führen. Der Rechenaufwand steigt aber stark an, besonders wenn zusätzlich zu Einzelwortmerkmalen der Wortkontext betrachtet wird.

Auch bei den Klassifikationsverfahren kann angesetzt werden. Es sollte überprüft werden, wie stark das Gesamtverfahren von Überanpassung betroffen ist. Vorstellbar ist, dass besonders bei kleinen Korpora mit ähnlichen Texten Überanpassung große Auswirkung auf das Ergebnis hat. Hierzu könnten die Verfahren auf einem der vorgestellten Korpora trainiert und auf anderen getestet werden.

Zu guter Letzt ließen sich die Ergebnisse der Klassifikationsverfahren ebenfalls zu kombinieren. Die dadurch zu erwartende Verbesserung erscheint aber minimal, und die mehrfache Kombination verstärkt wohl die Überanpassung.

Literaturverzeichnis

[AP96] Ali, Kamal M. ; Pazzani, Michael J.: Error reduction through learning multiple descriptions. In: *Machine Learning* 24 (1996), September, Nr. 3, S. 173–202. http://dx.doi.org/10.1007/BF00058611. – DOI 10.1007/BF00058611. – ISSN 0885–6125, 1573–0565 (Zitiert auf Seite 23).

[Apa15] Apache: *OpenNLP*. Oktober 2015 (Zitiert auf Seite 35).

[APK+00] Androutsopoulos, Ion ; Paliouras, Georgios ; Karkaletsis, Vangelis ; Sakkis, Georgios ; Spyropoulos, Constantine D. ; Stamatopoulos, Panagiotis: Learning to Filter Spam E-Mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. In: *arXiv:cs/0009009* (2000), September (Zitiert auf Seite 13).

[BKL09] Bird, Steven ; Klein, Ewan ; Loper, Edward: *Natural language processing with Python: [analyzing text with the natural language toolkit]*. 1. ed. Sebastopol, Calif. : O'Reilly, 2009. – ISBN 978–0–596–51649–9 (Zitiert auf Seite 31).

[BM00] Berthelsen, Harald ; Megyesi, Beáta: *Ensemble of Classifiers for Noise Detection in PoS Tagged Corpora*. 2000 (Zitiert auf Seite 26).

[BN12] Bohnet, Bernd ; Nivre, Joakim: A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2012 (EMNLP-CoNLL '12), S. 1455–1465 (Zitiert auf Seite 36).

[Bra00] Brants, Thorsten: TnT: A Statistical Part-of-speech Tagger. In: *Proceedings of the Sixth Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2000 (ANLC '00), S. 224–231 (Zitiert auf Seite 35).

[Bri92] Brill, Eric: A Simple Rule-based Part of Speech Tagger. In: *Proceedings of the Third Conference on Applied Natural Language Processing*. Stroudsburg, PA, USA : Association for Computational Linguistics, 1992 (ANLC '92), S. 152–155 (Zitiert auf Seite 37).

[BW98] Brill, Eric ; Wu, Jun: Classifier Combination for Improved Lexical Disambiguation. In: *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*. Stroudsburg, PA, USA : Association for Computational Linguistics, 1998 (COLING '98), S. 191–195 (Zitiert auf Seite 25).

[CAB+00] Conway, Matthew ; Audia, Steve ; Burnette, Tommy ; Cosgrove, Dennis ; Christiansen, Kevin: Alice: Lessons Learned from Building a 3D System

for Novices. In: *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*. The Hague, The Netherlands : ACM Press, 2000. – ISBN 1–58113–216–6, S. 486–493 (Zitiert auf den Seiten 1 und 31).

[CL01] Changchien, S. W. ; Lu, Tzu-Chuen: Mining association rules procedure to support on-line recommendation by customers and products fragmentation. In: *Expert Systems with Applications* 20 (2001), Mai, Nr. 4, S. 325–335. http://dx.doi.org/10.1016/S0957-4174(01)00017-3. – DOI 10.1016/S0957–4174(01)00017–3. – ISSN 0957–4174 (Zitiert auf Seite 16).

[Coo90] Cooper, Gregory F.: The computational complexity of probabilistic inference using bayesian belief networks. In: *Artificial Intelligence* 42 (1990), März, Nr. 2, S. 393–405. http://dx.doi.org/10.1016/0004-3702(90)90060-D. – DOI 10.1016/0004–3702(90)90060–D. – ISSN 0004–3702 (Zitiert auf Seite 14).

[CP12] Choi, Jinho D. ; Palmer, Martha: Fast and Robust Part-of-speech Tagging Using Dynamic Model Selection. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2012 (ACL '12), S. 363–367 (Zitiert auf Seite 35).

[CWB+11] Collobert, Ronan ; Weston, Jason ; Bottou, Léon ; Karlen, Michael ; Kavukcuoglu, Koray ; Kuksa, Pavel: Natural Language Processing (Almost) from Scratch. In: *J. Mach. Learn. Res.* 12 (2011), November, S. 2493–2537. – ISSN 1532–4435 (Zitiert auf Seite 36).

[de] de Kok, Daniel: *danieldk/jitar*. https://github.com/danieldk/jitar, (Zitiert auf Seite 35).

[DZBG96] Daelemans, Walter ; Zavrel, Jakub ; Berck, Peter ; Gillis, Steven: MBT: A Memory-Based Part of Speech Tagger-Generator. In: *arXiv:cmp-lg/9607012* (1996), Juli (Zitiert auf Seite 37).

[FBCC+10] Ferrucci, David ; Brown, Eric ; Chu-Carroll, Jennifer ; Fan, James ; Gondek, David ; Kalyanpur, Aditya A. ; Lally, Adam ; Murdock, J. W. ; Nyberg, Eric ; Prager, John ; Schlaefer, Nico ; Welty, Chris: Building Watson: An Overview of the DeepQA Project. In: *AI Magazine* 31 (2010), Juli, Nr. 3, S. 59–79. http://dx.doi.org/10.1609/aimag.v31i3.2303. – DOI 10.1609/aimag.v31i3.2303. – ISSN 0738–4602 (Zitiert auf den Seiten 2 und 24).

[FS99] Freund, Yoav ; Schapire, Robert E.: *A Short Introduction to Boosting*. 1999 (Zitiert auf Seite 24).

[GB05] Glass, Kevin ; Bangay, Shaun: Evaluating Parts-of-speech Taggers for Use in a Text-to-scene Conversion System. In: *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*. Republic of South Africa : South African Institute for Computer Scientists and Information Technologists, 2005 (SAICSIT '05). – ISBN 1–59593–258–5, S. 20–28 (Zitiert auf Seite 25).

[GM04] Giménez, Jesús ; Màrquez, Lluís: SVMTool: A general POS tagger generator based on Support Vector Machines. In: *In Proceedings of the 4th International Conference on Language Resources and Evaluation*, 2004 (Zitiert auf Seite 37).

[Ham12] Hampel, Sina: *Programmieren in natürlicher Sprache: Aufbau des Alice-Korpus*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Bachelorarbeit, November 2012 (Zitiert auf den Seiten 1 und 31).

[HDY+12] Hinton, G. ; Deng, L. ; Yu, D. ; Dahl, G. E. ; r Mohamed, A. ; Jaitly, N. ; Senior, A. ; Vanhoucke, V. ; Nguyen, P. ; Sainath, T. N. ; Kingsbury, B.: Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. In: *IEEE Signal Processing Magazine* 29 (2012), November, Nr. 6, S. 82–97. `http://dx.doi.org/10.1109/MSP.2012.2205597`. – DOI 10.1109/MSP.2012.2205597. – ISSN 1053–5888 (Zitiert auf Seite 19).

[Hos06] Hossain, Golam M.: *GPoSTTL*. 2006 (Zitiert auf Seite 37).

[IS04] Ide, Nancy ; Suderman, Keith: The American National Corpus First Release. In: *Proceedings of the Fourth Language Resources and Evaluation Conference (LREC*, 2004, S. 1681–1684 (Zitiert auf Seite 31).

[KA15] Khin, Nyein Pyae P. ; Aung, Than N.: Analyzing Tagging Accuracy of Part-Of-Speech Taggers. In: *Genetic and Evolutionary Computing*. Springer, 2015, S. 347–354 (Zitiert auf Seite 26).

[KF67] Kučera, Henry ; Francis, Nelson: *Brown Corpus*. `https://en.wikipedia.org/wiki/Brown_Corpus`, 1967. – Page Version ID: 686472788 (Zitiert auf Seite 37).

[Koh96] Kohavi, Ron: Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid. In: *PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING*, AAAI Press, 1996, S. 202–207 (Zitiert auf Seite 15).

[LMP01] Lafferty, John ; McCallum, Andrew ; Pereira, Fernando: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: *Departmental Papers (CIS)* (2001), Juni (Zitiert auf Seite 37).

[Mac] Maciej Piasecki, Adam W.: Multiclassifier Approach to Tagging of Polish. (Zitiert auf Seite 27).

[Man11] Manning, Christopher D.: Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In: Gelbukh, Alexander F. (Hrsg.): *Computational Linguistics and Intelligent Text Processing*. Springer Berlin Heidelberg, 2011 (Lecture Notes in Computer Science 6608). – ISBN 978–3–642–19399–6 978–3–642–19400–9, S. 171–189 (Zitiert auf Seite 6).

[MB90] Morgan, N. ; Bourlard, H.: Continuous speech recognition using multilayer perceptrons with hidden Markov models. In: *, 1990 International Conference on Acoustics, Speech, and Signal Processing, 1990. ICASSP-90*, 1990, S. 413–416 vol.1 (Zitiert auf Seite 19).

[MCS99] Manna, S. L. ; Colia, A. M. ; Sperduti, A.: Optical font recognition for multi-font OCR and document processing. In: *Tenth International Workshop on Database and Expert Systems Applications, 1999. Proceedings*, 1999, S. 549–553 (Zitiert auf Seite 20).

[MMS93] Marcus, Mitchell P. ; Marcinkiewicz, Mary A. ; Santorini, Beatrice: Building a Large Annotated Corpus of English: The Penn Treebank. In: *Comput. Linguist.* 19 (1993), Juni, Nr. 2, S. 313–330. – ISSN 0891–2017 (Zitiert auf den Seiten xv, 1, 32 und 33).

[MSB93] Mazroua, A. A. ; Salama, M. M. A. ; Bartnikas, R.: PD pattern recognition with neural networks using the multilayer perceptron technique. In: *IEEE Transactions on Electrical Insulation* 28 (1993), Dezember, Nr. 6, S. 1082–1089. http://dx.doi.org/10.1109/14.249382. – DOI 10.1109/14.249382. – ISSN 0018–9367 (Zitiert auf Seite 19).

[MSP05] Molinaro, A. M. ; Simon, R. ; Pfeiffer, R. M.: Prediction error estimation: a comparison of resampling methods. In: *Bioinformatics* 21 (2005), August, Nr. 15, S. 3301–3307. http://dx.doi.org/10.1093/bioinformatics/bti499. – DOI 10.1093/bioinformatics/bti499. – ISSN 1367–4803, 1460–2059 (Zitiert auf Seite 21).

[PBTK06] Petrov, Slav ; Barrett, Leon ; Thibaux, Romain ; Klein, Dan: Learning Accurate, Compact, and Interpretable Tree Annotation. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2006 (ACL-44), S. 433–440 (Zitiert auf Seite 36).

[Ped00] Pedersen, Ted: A Simple Approach to Building Ensembles of Naive Bayesian Classifiers for Word Sense Disambiguation. In: *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2000 (NAACL 2000), S. 63–69 (Zitiert auf Seite 24).

[PG07] Polat, Kemal ; Güneş, Salih: Classification of epileptiform EEG using a hybrid system based on decision tree classifier and fast Fourier transform. In: *Applied Mathematics and Computation* 187 (2007), April, Nr. 2, S. 1017–1026. http://dx.doi.org/10.1016/j.amc.2006.09.022. – DOI 10.1016/j.amc.2006.09.022. – ISSN 0096–3003 (Zitiert auf Seite 15).

[Pha06] Phan, Xuan-Hieu: Crftagger: Crf english pos tagger. In: *URL: http://crftagger. sourceforge. net* (2006) (Zitiert auf Seite 37).

[Rac] Rachel V Xavier Aires, Sandra M. A.: Combining Multiple Classifiers to Improve Part of Speech Tagging: A Case Study for Brazilian Portuguese. (Zitiert auf Seite 27).

[RZ98] Roth, Dan ; Zelenko, Dmitry: Part of Speech Tagging Using a Network of Linear Separators. In: *Proceedings of the 17th International Conference on Computational Linguistics - Volume 2*. Stroudsburg, PA, USA : Association for Computational Linguistics, 1998 (COLING '98), S. 1136–1142 (Zitiert auf Seite 35).

[Sav01] Savary, Agata: Typographical Nearest-Neighbor Search in a Finite-State Lexicon and Its Application to Spelling Correction. In: Watson, Bruce W. (Hrsg.) ; Wood, Derick (Hrsg.): *Implementation and Application of Automata*. Springer Berlin Heidelberg, Juli 2001 (Lecture Notes in Computer Science 2494). – ISBN 978–3–540–00400–4 978–3–540–36390–3, S. 251–260 (Zitiert auf Seite 20).

[Sch94] Schmid, Helmut: *Probabilistic Part-of-Speech Tagging Using Decision Trees*. 1994 (Zitiert auf Seite 35).

[SCJ07] Shen, Libin ; Champollion, Lucas ; Joshi, Aravind K.: LTAG-spinal and the Treebank. In: *Language Resources and Evaluation* 42 (2007), Oktober, Nr. 1, S. 1–19. http://dx.doi.org/10.1007/s10579-007-9043-7. – DOI

10.1007/s10579–007–9043–7. – ISSN 1574–020X, 1572–8412 (Zitiert auf Seite 37).

[Sjö03] Sjöbergh, Jonas: Combining POS-taggers for improved accuracy on Swedish text. (2003) (Zitiert auf Seite 27).

[Søg] Søgaard, Anders: Ensemble-based POS tagging of Italian. (Zitiert auf Seite 27).

[TKMS03] Toutanova, Kristina ; Klein, Dan ; Manning, Christopher D. ; Singer, Yoram: Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network. In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2003 (NAACL '03), S. 173–180 (Zitiert auf Seite 34).

[TMi11] Tsuruoka, Yoshimasa ; Miyao, Yusuke ; 'ichi Kazama, Jun: Learning with Lookahead: Can History-based Models Rival Globally Optimized Models? In: *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*. Stroudsburg, PA, USA : Association for Computational Linguistics, 2011 (CoNLL '11). – ISBN 978–1–932432–92–3, S. 238–246 (Zitiert auf Seite 38).

[VJ01] Viola, Paul ; Jones, Michael: *Rapid object detection using a boosted cascade of simple features*. 2001 (Zitiert auf Seite 12).

[vZD98] van Halteren, Hans ; Zavrel, Jakub ; Daelemans, Walter: Improving Data Driven Wordclass Tagging by System Combination. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*. Stroudsburg, PA, USA : Association for Computational Linguistics, 1998 (ACL '98), S. 491–497 (Zitiert auf Seite 25).

[vZD01] van Halteren, Hans ; Zavrel, Jakub ; Daelemans, Walter: Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems. In: *Computational Linguistics* 27 (2001), Juni, Nr. 2, S. 199–229. `http://dx.doi.org/10.1162/089120101750300508`. – DOI 10.1162/089120101750300508. – ISSN 0891–2017 (Zitiert auf Seite 25).

[WFH11] Witten, Ian H. ; Frank, Eibe ; Hall, Mark A.: *Data Mining: Practical Machine Learning Tools and Techniques: Practical Machine Learning Tools and Techniques*. Elsevier, 2011. – ISBN 978–0–08–089036–4 (Zitiert auf den Seiten 8 und 20).

[WT15] Weigelt, S. ; Tichy, W. F.: Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)* Bd. 2, 2015, S. 819–820 (Zitiert auf den Seiten 2 und 33).

[ZC05] Zou, Min ; Conzen, Suzanne D.: A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. In: *Bioinformatics* 21 (2005), Januar, Nr. 1, S. 71–79. `http://dx.doi.org/10.1093/bioinformatics/bth463`. – DOI 10.1093/bioinformatics/bth463. – ISSN 1367–4803, 1460–2059 (Zitiert auf Seite 14).

Anhang

A. Beispiel-Datensatz für die Wetterklassifikation

Eingabedaten			Klassifikation
Regen	Temperatur	Nebel	Schnee
ja	kalt	ja	ja
nein	kalt	nein	ja
nein	kalt	ja	ja
ja	kalt	ja	ja
nein	heiß	ja	nein
ja	heiß	nein	nein
ja	heiß	ja	ja
ja	kalt	nein	nein
ja	heiß	ja	nein
ja	kalt	nein	nein
nein	kalt	ja	nein
ja	kalt	nein	ja
nein	heiß	nein	ja
nein	heiß	ja	nein
nein	kalt	ja	nein
ja	kalt	nein	ja
ja	kalt	ja	ja
ja	heiß	ja	ja
nein	kalt	ja	nein
nein	kalt	nein	nein
nein	kalt	nein	nein
nein	kalt	nein	ja
nein	kalt	nein	ja
ja	kalt	nein	nein
nein	heiß	ja	ja
nein	kalt	ja	nein
nein	kalt	ja	nein
nein	heiß	nein	nein
nein	heiß	ja	nein
nein	heiß	nein	nein

B. Tabellearische Darstellung der Lernkurven

B.1. Lernkurve für WSJ-Korpus mit Wörtern

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	97,8%	0,0252	0,9764	37,085s
HyperPipes	1,0%	93,78%	0,0745	0,9334	5,8s
IBk	1,0%	97,69%	0,0258	0,9752	48,147s
J48	1,0%	97,77%	0,0244	0,9761	2,927s
LibSVM	1,0%	15,03%	0,1642	0,0126	260,354s
NaiveBayes	1,0%	97,58%	0,0261	0,974	56,546s
OneR	1,0%	78,87%	0,0689	0,7706	1,595s
PART	1,0%	97,45%	0,0256	0,9726	4,724s
REPTree	1,0%	65,76%	0,0794	0,6203	3,521s
VFI	1,0%	94,75%	0,0342	0,944	17,625s
NNge	1,0%	93,29%	0,0427	0,9282	52,209s
JRip	1,0%	96,7%	0,0318	0,9645	7,718s
KStar	1,0%	97,85%	0,025	0,9769	1514,119s
Ridor	1,0%	93,43%	0,0456	0,9295	20,201s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	98,04%	0,0243	0,979	34,349s
HyperPipes	2,0%	91,81%	0,0855	0,9125	4,505s
IBk	2,0%	98,02%	0,0242	0,9788	90,165s
J48	2,0%	98,33%	0,0222	0,9821	3,273s
LibSVM	2,0%	13,99%	0,1652	0,0	398,564s
NaiveBayes	2,0%	97,99%	0,0245	0,9784	57,953s
OneR	2,0%	88,54%	0,0471	0,8759	1,36s
PART	2,0%	97,85%	0,0238	0,977	6,836s
REPTree	2,0%	70,5%	0,074	0,6747	6,272s
VFI	2,0%	95,8%	0,0354	0,9551	18,614s
NNge	2,0%	95,63%	0,0362	0,9531	94,474s
JRip	2,0%	97,43%	0,0281	0,9724	13,632s
KStar	2,0%	98,09%	0,0235	0,9795	2937,237s
Ridor	2,0%	95,65%	0,0371	0,9533	32,67s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	98,15%	0,0238	0,9802	37,671s
HyperPipes	5,0%	82,35%	0,0951	0,8112	4,998s
IBk	5,0%	98,24%	0,0227	0,9812	190,565s
J48	5,0%	98,52%	0,0209	0,9841	4,916s
LibSVM	5,0%	48,24%	0,1281	0,4094	851,422s
NaiveBayes	5,0%	98,14%	0,0239	0,98	56,114s
OneR	5,0%	98,5%	0,0219	0,9839	1,494s
PART	5,0%	98,33%	0,0217	0,982	11,596s
REPTree	5,0%	74,35%	0,0715	0,7191	17,679s
VFI	5,0%	94,16%	0,0387	0,9377	20,879s
NNge	5,0%	95,9%	0,0353	0,9559	204,151s
JRip	5,0%	98,14%	0,0238	0,98	32,385s
KStar	5,0%	98,28%	0,0223	0,9816	7399,434s
Ridor	5,0%	96,85%	0,0316	0,9663	79,998s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	98,2%	0,0234	0,9806	31,212s
HyperPipes	10,0%	82,46%	0,0997	0,8117	4,358s
IBk	10,0%	98,32%	0,0221	0,982	391,238s
J48	10,0%	98,6%	0,0204	0,985	7,043s
LibSVM	10,0%	68,53%	0,0999	0,6491	1494,897s
NaiveBayes	10,0%	98,19%	0,0236	0,9805	57,137s
OneR	10,0%	98,52%	0,0216	0,9842	1,287s
PART	10,0%	98,48%	0,0206	0,9837	20,364s
REPTree	10,0%	79,33%	0,0644	0,7748	32,079s
VFI	10,0%	92,87%	0,0422	0,9238	21,289s
NNge	10,0%	94,37%	0,0412	0,9395	395,557s
JRip	10,0%	98,33%	0,0225	0,982	74,016s
KStar	10,0%	98,35%	0,0215	0,9823	14120,456s
Ridor	10,0%	97,42%	0,0286	0,9723	173,777s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	98,26%	0,0231	0,9813	29,072s
HyperPipes	20,0%	76,97%	0,1042	0,7529	4,503s
IBk	20,0%	98,39%	0,0215	0,9827	661,661s
J48	20,0%	98,66%	0,0199	0,9856	10,924s
LibSVM	20,0%	87,17%	0,0638	0,8613	2498,64s
NaiveBayes	20,0%	98,21%	0,0234	0,9808	45,657s
OneR	20,0%	98,55%	0,0214	0,9845	0,945s
PART	20,0%	98,55%	0,0202	0,9844	23,89s
REPTree	20,0%	83,88%	0,0573	0,825	70,684s
VFI	20,0%	93,99%	0,0413	0,9358	19,913s
NNge	20,0%	96,7%	0,0322	0,9646	877,132s
JRip	20,0%	98,5%	0,0211	0,9839	185,844s
KStar	20,0%	98,44%	0,0209	0,9833	25119,136s
Ridor	20,0%	97,83%	0,0263	0,9767	505,374s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	98,27%	0,023	0,9814	24,647s
HyperPipes	30,0%	76,07%	0,1065	0,7428	3,666s
IBk	30,0%	98,46%	0,0208	0,9835	803,53s
J48	30,0%	98,69%	0,0196	0,986	15,281s
LibSVM	30,0%	89,88%	0,0567	0,8908	3444,768s
NaiveBayes	30,0%	98,23%	0,0232	0,981	39,804s
OneR	30,0%	98,55%	0,0215	0,9844	0,946s
PART	30,0%	98,63%	0,0195	0,9853	82,084s
REPTree	30,0%	88,78%	0,0464	0,8785	112,144s
VFI	30,0%	94,6%	0,0421	0,9422	16,297s
NNge	30,0%	96,92%	0,031	0,9669	1264,959s
JRip	30,0%	98,6%	0,0203	0,9849	330,939s
KStar	30,0%	98,46%	0,0206	0,9835	33128,341s
Ridor	30,0%	97,99%	0,0252	0,9785	1062,62s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	98,24%	0,0232	0,9812	21,777s
HyperPipes	40,0%	75,07%	0,108	0,7321	3,179s
IBk	40,0%	98,49%	0,0207	0,9838	1229,51s
J48	40,0%	98,71%	0,0194	0,9862	18,224s
LibSVM	40,0%	90,93%	0,0536	0,9022	4367,151s
NaiveBayes	40,0%	98,24%	0,0232	0,9811	34,162s
OneR	40,0%	98,53%	0,0216	0,9842	0,866s
PART	40,0%	98,66%	0,0193	0,9857	90,106s
REPTree	40,0%	93,18%	0,0353	0,9263	129,829s
VFI	40,0%	94,82%	0,0412	0,9446	14,04s
NNge	40,0%	97,2%	0,0294	0,9699	1977,45s
JRip	40,0%	98,62%	0,0201	0,9852	570,115s
KStar	40,0%	98,51%	0,0204	0,984	37927,529s
Ridor	40,0%	98,16%	0,0242	0,9803	2162,98s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	98,26%	0,0231	0,9813	18,511s
HyperPipes	50,0%	74,83%	0,1089	0,7298	2,725s
IBk	50,0%	98,53%	0,0204	0,9843	1521,835s
J48	50,0%	98,73%	0,0191	0,9863	22,941s
LibSVM	50,0%	93,53%	0,0452	0,9304	5080,381s
NaiveBayes	50,0%	98,25%	0,0232	0,9812	32,109s
OneR	50,0%	98,55%	0,0214	0,9845	0,969s
PART	50,0%	98,63%	0,0193	0,9853	222,018s
REPTree	50,0%	98,64%	0,0202	0,9854	74,504s
VFI	50,0%	95,11%	0,0416	0,9477	12,143s
NNge	50,0%	97,65%	0,027	0,9748	2899,129s
JRip	50,0%	98,66%	0,0199	0,9856	907,451s
KStar	50,0%	98,52%	0,0202	0,9841	39810,77s
Ridor	50,0%	98,2%	0,0239	0,9807	4377,822s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	98,23%	0,0232	0,981	14,744s
HyperPipes	60,0%	74,39%	0,1098	0,7248	2,23s
IBk	60,0%	98,53%	0,0202	0,9842	1697,582s
J48	60,0%	98,73%	0,0191	0,9864	24,946s
LibSVM	60,0%	95,28%	0,0387	0,9492	5250,991s
NaiveBayes	60,0%	98,23%	0,0232	0,981	22,789s
OneR	60,0%	98,55%	0,0214	0,9845	0,803s
PART	60,0%	98,66%	0,0191	0,9856	268,747s
REPTree	60,0%	98,68%	0,0198	0,9858	85,976s
VFI	60,0%	95,35%	0,0409	0,9502	9,83s
NNge	60,0%	97,89%	0,0258	0,9773	3575,478s
JRip	60,0%	98,67%	0,0197	0,9858	1360,802s
KStar	60,0%	98,55%	0,0199	0,9845	38188,696s
Ridor	60,0%	98,26%	0,0235	0,9813	8502,903s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	98,24%	0,0231	0,9811	11,273s
HyperPipes	70,0%	73,33%	0,1105	0,7131	1,828s
IBk	70,0%	98,55%	0,0201	0,9845	1792,307s
J48	70,0%	98,71%	0,0191	0,9862	28,879s
LibSVM	70,0%	95,56%	0,0375	0,9522	5543,179s
NaiveBayes	70,0%	98,29%	0,0228	0,9817	19,498s
OneR	70,0%	98,54%	0,0216	0,9843	0,823s
PART	70,0%	98,7%	0,0187	0,9861	283,461s
REPTree	70,0%	98,68%	0,0198	0,9859	106,732s
VFI	70,0%	95,61%	0,0407	0,953	7,632s
NNge	70,0%	97,21%	0,0295	0,9701	4808,647s
JRip	70,0%	98,68%	0,0195	0,9858	1994,431s
KStar	70,0%	98,51%	0,0201	0,984	33041,927s
Ridor	70,0%	98,32%	0,0231	0,982	13074,183s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	98,22%	0,0232	0,981	8,937s
HyperPipes	80,0%	72,74%	0,1112	0,7068	1,414s
IBk	80,0%	98,56%	0,0199	0,9846	1319,258s
J48	80,0%	98,75%	0,0187	0,9866	29,121s
LibSVM	80,0%	95,83%	0,0364	0,9551	5719,75s
NaiveBayes	80,0%	98,25%	0,0231	0,9812	11,559s
OneR	80,0%	98,55%	0,0214	0,9845	0,752s
PART	80,0%	98,76%	0,0184	0,9867	393,288s
REPTree	80,0%	98,69%	0,0197	0,9859	121,369s
VFI	80,0%	95,69%	0,0406	0,9539	5,184s
NNge	80,0%	97,65%	0,027	0,9748	5791,72s
JRip	80,0%	98,69%	0,0195	0,9859	2930,838s
KStar	80,0%	98,51%	0,0201	0,984	25278,705s
Ridor	80,0%	98,34%	0,0229	0,9822	17632,267s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	98,25%	0,0231	0,9812	5,402s
HyperPipes	90,0%	71,65%	0,1115	0,6951	0,856s
IBk	90,0%	98,54%	0,02	0,9843	814,028s
J48	90,0%	98,85%	0,0183	0,9876	34,594s
LibSVM	90,0%	95,91%	0,036	0,956	5765,173s
NaiveBayes	90,0%	98,2%	0,0234	0,9807	6,007s
OneR	90,0%	98,53%	0,0216	0,9842	0,716s
PART	90,0%	98,7%	0,0187	0,986	546,922s
REPTree	90,0%	98,69%	0,0196	0,9859	130,708s
VFI	90,0%	95,8%	0,0409	0,955	3,042s
NNge	90,0%	98,05%	0,0247	0,9791	7930,216s
JRip	90,0%	98,62%	0,0201	0,9851	4138,958s
KStar	90,0%	98,5%	0,0201	0,9839	14635,953s
Ridor	90,0%	98,36%	0,0228	0,9824	27795,116s

B.2. Lernkurve für WSJ-Korpus ohne Wörtern

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	97,81%	0,0253	0,9765	32,32s
HyperPipes	1,0%	92,17%	0,0777	0,9163	4,387s
IBk	1,0%	97,88%	0,0251	0,9772	50,209s
J48	1,0%	98,04%	0,0236	0,9789	2,247s
LibSVM	1,0%	86,3%	0,0658	0,8515	128,991s
NaiveBayes	1,0%	97,83%	0,0251	0,9768	55,626s
OneR	1,0%	98,02%	0,025	0,9787	1,341s
PART	1,0%	97,2%	0,0264	0,9699	2,547s
REPTree	1,0%	97,82%	0,0251	0,9766	1,353s
VFI	1,0%	96,35%	0,0321	0,9611	18,067s
NNge	1,0%	97,71%	0,027	0,9754	7,046s
JRip	1,0%	96,76%	0,0316	0,9651	1,829s
KStar	1,0%	97,84%	0,0247	0,9769	1368,287s
Ridor	1,0%	93,11%	0,0466	0,926	2,892s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	98,06%	0,0242	0,9792	29,437s
HyperPipes	2,0%	84,04%	0,0864	0,8296	4,053s
IBk	2,0%	98,08%	0,0239	0,9794	91,705s
J48	2,0%	98,4%	0,0217	0,9828	2,256s
LibSVM	2,0%	92,25%	0,0496	0,9164	159,344s
NaiveBayes	2,0%	98,03%	0,0244	0,9789	50,046s
OneR	2,0%	98,38%	0,0226	0,9826	1,098s
PART	2,0%	97,86%	0,0237	0,977	2,506s
REPTree	2,0%	98,26%	0,0226	0,9814	1,152s
VFI	2,0%	94,76%	0,036	0,9443	18,129s
NNge	2,0%	97,97%	0,0253	0,9783	9,953s
JRip	2,0%	97,51%	0,0277	0,9733	2,713s
KStar	2,0%	98,12%	0,0233	0,9799	2713,459s
Ridor	2,0%	95,57%	0,0375	0,9525	5,049s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	98,15%	0,0238	0,9802	31,504s
HyperPipes	5,0%	71,31%	0,0956	0,6935	5,219s
IBk	5,0%	98,27%	0,0226	0,9814	231,183s
J48	5,0%	98,5%	0,021	0,9839	2,421s
LibSVM	5,0%	96,28%	0,0343	0,96	260,456s
NaiveBayes	5,0%	98,16%	0,0237	0,9803	48,503s
OneR	5,0%	98,5%	0,0218	0,984	1,098s
PART	5,0%	98,35%	0,0215	0,9823	2,925s
REPTree	5,0%	98,5%	0,0212	0,9839	1,276s
VFI	5,0%	92,1%	0,0406	0,916	17,324s
NNge	5,0%	97,9%	0,0254	0,9775	18,712s
JRip	5,0%	98,15%	0,0237	0,9802	5,119s
KStar	5,0%	98,27%	0,0222	0,9814	6660,905s
Ridor	5,0%	96,93%	0,0312	0,9671	19,764s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	98,21%	0,0234	0,9808	32,087s
HyperPipes	10,0%	61,29%	0,101	0,5858	5,575s
IBk	10,0%	98,36%	0,0218	0,9824	399,338s
J48	10,0%	98,56%	0,0205	0,9846	2,154s
LibSVM	10,0%	97,06%	0,0306	0,9684	316,181s
NaiveBayes	10,0%	98,19%	0,0236	0,9806	45,851s
OneR	10,0%	98,51%	0,0217	0,9841	1,007s
PART	10,0%	98,44%	0,0208	0,9832	3,28s
REPTree	10,0%	98,54%	0,0208	0,9843	1,424s
VFI	10,0%	89,17%	0,0467	0,8851	17,405s
NNge	10,0%	98,26%	0,0235	0,9814	39,225s
JRip	10,0%	98,29%	0,0228	0,9817	10,602s
KStar	10,0%	98,36%	0,0215	0,9824	12829,086s
Ridor	10,0%	97,54%	0,0279	0,9736	38,953s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	98,25%	0,0231	0,9812	26,904s
HyperPipes	20,0%	47,65%	0,1052	0,4381	4,146s
IBk	20,0%	98,41%	0,0212	0,983	645,619s
J48	20,0%	98,62%	0,02	0,9852	2,392s
LibSVM	20,0%	97,8%	0,0264	0,9763	432,369s
NaiveBayes	20,0%	98,22%	0,0233	0,9809	40,854s
OneR	20,0%	98,54%	0,0216	0,9843	1,028s
PART	20,0%	98,55%	0,02	0,9844	4,481s
REPTree	20,0%	98,58%	0,0202	0,9848	2,322s
VFI	20,0%	88,23%	0,052	0,8751	17,86s
NNge	20,0%	98,05%	0,0245	0,9791	88,008s
JRip	20,0%	98,48%	0,0213	0,9837	27,877s
KStar	20,0%	98,42%	0,021	0,9831	22939,267s
Ridor	20,0%	97,79%	0,0265	0,9762	73,848s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	98,23%	0,0232	0,981	20,972s
HyperPipes	30,0%	39,91%	0,1075	0,3546	3,385s
IBk	30,0%	98,51%	0,0206	0,984	976,39s
J48	30,0%	98,67%	0,0197	0,9857	2,76s
LibSVM	30,0%	98,16%	0,0242	0,9802	560,159s
NaiveBayes	30,0%	98,23%	0,0232	0,981	35,7s
OneR	30,0%	98,54%	0,0215	0,9844	0,968s
PART	30,0%	98,6%	0,0196	0,985	5,996s
REPTree	30,0%	98,62%	0,0198	0,9852	3,823s
VFI	30,0%	88,21%	0,054	0,8749	14,443s
NNge	30,0%	98,17%	0,0239	0,9804	120,205s
JRip	30,0%	98,5%	0,0211	0,984	58,739s
KStar	30,0%	98,43%	0,0208	0,9832	30332,07s
Ridor	30,0%	98,04%	0,0249	0,979	134,606s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	98,25%	0,0231	0,9813	20,607s
HyperPipes	40,0%	38,08%	0,1089	0,3316	3,339s
IBk	40,0%	98,48%	0,0206	0,9837	1604,199s
J48	40,0%	98,69%	0,0193	0,986	3,621s
LibSVM	40,0%	98,23%	0,0237	0,981	707,099s
NaiveBayes	40,0%	98,23%	0,0232	0,981	30,656s
OneR	40,0%	98,55%	0,0215	0,9844	0,983s
PART	40,0%	98,63%	0,0193	0,9853	7,827s
REPTree	40,0%	98,66%	0,0194	0,9856	4,357s
VFI	40,0%	88,17%	0,0582	0,8745	14,267s
NNge	40,0%	98,46%	0,0221	0,9835	179,087s
JRip	40,0%	98,54%	0,0208	0,9844	103,632s
KStar	40,0%	98,46%	0,0205	0,9835	34777,1s
Ridor	40,0%	98,1%	0,0245	0,9797	223,522s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	98,25%	0,0231	0,9812	17,402s
HyperPipes	50,0%	36,52%	0,1099	0,3145	2,577s
IBk	50,0%	98,54%	0,0202	0,9843	1697,68s
J48	50,0%	98,72%	0,0191	0,9862	3,168s
LibSVM	50,0%	98,26%	0,0235	0,9813	860,471s
NaiveBayes	50,0%	98,25%	0,023	0,9813	28,028s
OneR	50,0%	98,52%	0,0217	0,9841	0,846s
PART	50,0%	98,64%	0,0191	0,9854	8,535s
REPTree	50,0%	98,64%	0,0194	0,9854	4,416s
VFI	50,0%	88,21%	0,0597	0,875	11,045s
NNge	50,0%	98,46%	0,0221	0,9835	207,756s
JRip	50,0%	98,54%	0,0207	0,9843	156,316s
KStar	50,0%	98,45%	0,0205	0,9834	35881,858s
Ridor	50,0%	98,16%	0,0242	0,9802	323,21s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	98,27%	0,0229	0,9814	13,406s
HyperPipes	60,0%	35,21%	0,1107	0,3	2,201s
IBk	60,0%	98,55%	0,02	0,9845	2094,916s
J48	60,0%	98,7%	0,0191	0,986	4,574s
LibSVM	60,0%	98,29%	0,0233	0,9816	976,453s
NaiveBayes	60,0%	98,24%	0,0232	0,9812	22,635s
OneR	60,0%	98,53%	0,0216	0,9842	1,58s
PART	60,0%	98,64%	0,0191	0,9854	13,181s
REPTree	60,0%	98,66%	0,0191	0,9856	8,968s
VFI	60,0%	88,2%	0,0599	0,8748	9,943s
NNge	60,0%	98,37%	0,0228	0,9825	268,974s
JRip	60,0%	98,57%	0,0205	0,9847	223,79s
KStar	60,0%	98,45%	0,0205	0,9834	34747,717s
Ridor	60,0%	98,24%	0,0236	0,9811	481,879s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	98,25%	0,0231	0,9812	10,819s
HyperPipes	70,0%	33,13%	0,1115	0,2757	1,733s
IBk	70,0%	98,58%	0,0198	0,9848	2056,312s
J48	70,0%	98,71%	0,019	0,9862	5,035s
LibSVM	70,0%	98,31%	0,0232	0,9818	1088,488s
NaiveBayes	70,0%	98,24%	0,0232	0,9811	15,539s
OneR	70,0%	98,55%	0,0215	0,9844	1,312s
PART	70,0%	98,66%	0,0189	0,9856	15,69s
REPTree	70,0%	98,7%	0,0189	0,9861	7,779s
VFI	70,0%	88,09%	0,061	0,8737	6,995s
NNge	70,0%	98,41%	0,0224	0,983	288,958s
JRip	70,0%	98,59%	0,0203	0,9848	298,279s
KStar	70,0%	98,46%	0,0204	0,9835	30524,833s
Ridor	70,0%	98,27%	0,0234	0,9815	583,542s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	98,24%	0,0232	0,9811	6,781s
HyperPipes	80,0%	32,94%	0,112	0,2738	1,208s
IBk	80,0%	98,55%	0,0201	0,9844	1544,497s
J48	80,0%	98,7%	0,019	0,9861	5,307s
LibSVM	80,0%	98,36%	0,0228	0,9824	1219,754s
NaiveBayes	80,0%	98,28%	0,0229	0,9816	10,475s
OneR	80,0%	98,52%	0,0217	0,9841	1,311s
PART	80,0%	98,7%	0,0187	0,986	18,177s
REPTree	80,0%	98,64%	0,0193	0,9854	8,835s
VFI	80,0%	88,17%	0,0621	0,8745	4,838s
NNge	80,0%	98,4%	0,0225	0,9828	318,752s
JRip	80,0%	98,62%	0,0201	0,9852	378,687s
KStar	80,0%	98,5%	0,0203	0,9839	23122,164s
Ridor	80,0%	98,28%	0,0233	0,9816	747,568s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	98,22%	0,0232	0,9809	3,972s
HyperPipes	90,0%	32,33%	0,1126	0,2664	0,739s
IBk	90,0%	98,53%	0,0201	0,9843	817,303s
J48	90,0%	98,73%	0,0188	0,9864	4,737s
LibSVM	90,0%	98,4%	0,0225	0,9828	1059,219s
NaiveBayes	90,0%	98,25%	0,0231	0,9813	5,942s
OneR	90,0%	98,52%	0,0216	0,9842	0,676s
PART	90,0%	98,64%	0,019	0,9854	16,552s
REPTree	90,0%	98,7%	0,0186	0,9861	9,137s
VFI	90,0%	88,27%	0,0623	0,8755	2,579s
NNge	90,0%	98,44%	0,0222	0,9832	351,957s
JRip	90,0%	98,61%	0,0202	0,985	453,969s
KStar	90,0%	98,47%	0,0203	0,9836	12906,791s
Ridor	90,0%	98,28%	0,0234	0,9815	901,489s

B.3. Lernkurve für NLCI-Korpus mit Wörtern

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	94,54%	0,0457	0,9387	3,471s
HyperPipes	1,0%	95,03%	0,0752	0,9439	0,49s
IBk	1,0%	95,77%	0,0446	0,9524	2,213s
J48	1,0%	93,65%	0,0519	0,9283	0,248s
LibSVM	1,0%	37,44%	0,1836	0,2265	10,514s
NaiveBayes	1,0%	94,53%	0,0464	0,9385	6,218s
OneR	1,0%	68,0%	0,1315	0,6276	0,196s
PART	1,0%	89,8%	0,0617	0,8854	0,322s
REPTree	1,0%	69,15%	0,0931	0,6407	0,196s
VFI	1,0%	94,63%	0,0509	0,9398	2,11s
NNge	1,0%	93,84%	0,0562	0,9308	1,131s
JRip	1,0%	86,36%	0,08	0,8435	0,281s
KStar	1,0%	95,06%	0,0472	0,9443	53,256s
Ridor	1,0%	80,41%	0,1026	0,7799	0,283s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	96,87%	0,0377	0,9649	3,804s
HyperPipes	2,0%	96,56%	0,079	0,9613	0,546s
IBk	2,0%	96,89%	0,0383	0,965	4,114s
J48	2,0%	95,37%	0,0444	0,9479	0,284s
LibSVM	2,0%	60,7%	0,1454	0,5289	17,291s
NaiveBayes	2,0%	96,54%	0,0391	0,9612	6,837s
OneR	2,0%	75,87%	0,1142	0,7215	0,177s
PART	2,0%	93,85%	0,0498	0,931	0,39s
REPTree	2,0%	79,2%	0,0766	0,7603	0,241s
VFI	2,0%	96,35%	0,0459	0,9591	2,228s
NNge	2,0%	96,49%	0,0425	0,9606	1,508s
JRip	2,0%	93,39%	0,0578	0,9252	0,411s
KStar	2,0%	96,61%	0,0395	0,9619	110,182s
Ridor	2,0%	88,49%	0,0785	0,8705	0,499s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	97,44%	0,0351	0,9713	3,667s
HyperPipes	5,0%	96,72%	0,0885	0,9632	0,544s
IBk	5,0%	97,87%	0,0322	0,9761	8,114s
J48	5,0%	97,06%	0,0369	0,9669	0,298s
LibSVM	5,0%	71,06%	0,125	0,6627	27,723s
NaiveBayes	5,0%	97,32%	0,0361	0,9699	6,577s
OneR	5,0%	85,67%	0,088	0,8367	0,166s
PART	5,0%	96,5%	0,0392	0,9607	0,501s
REPTree	5,0%	86,93%	0,0622	0,8512	0,278s
VFI	5,0%	97,29%	0,0391	0,9697	2,6s
NNge	5,0%	96,98%	0,0398	0,9661	2,909s
JRip	5,0%	96,35%	0,0434	0,9589	0,825s
KStar	5,0%	97,64%	0,0332	0,9735	266,845s
Ridor	5,0%	94,66%	0,0536	0,94	1,231s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	97,71%	0,0337	0,9743	3,158s
HyperPipes	10,0%	96,02%	0,0961	0,9551	0,487s
IBk	10,0%	98,3%	0,0288	0,9809	13,24s
J48	10,0%	97,76%	0,0322	0,9749	0,291s
LibSVM	10,0%	86,35%	0,0858	0,8442	39,279s
NaiveBayes	10,0%	97,62%	0,0342	0,9733	5,664s
OneR	10,0%	90,35%	0,0722	0,8908	0,147s
PART	10,0%	97,29%	0,0344	0,9695	0,584s
REPTree	10,0%	91,15%	0,0512	0,8997	0,348s
VFI	10,0%	97,93%	0,0347	0,9768	2,442s
NNge	10,0%	97,62%	0,0355	0,9733	4,65s
JRip	10,0%	97,11%	0,0387	0,9676	1,591s
KStar	10,0%	98,25%	0,0285	0,9804	502,175s
Ridor	10,0%	95,83%	0,0474	0,9533	2,552s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	97,95%	0,032	0,977	2,819s
HyperPipes	20,0%	95,84%	0,1038	0,9531	0,435s
IBk	20,0%	98,63%	0,026	0,9847	21,697s
J48	20,0%	98,26%	0,0289	0,9804	0,321s
LibSVM	20,0%	91,54%	0,0676	0,9041	55,682s
NaiveBayes	20,0%	97,83%	0,033	0,9757	5,043s
OneR	20,0%	93,18%	0,0607	0,9231	0,134s
PART	20,0%	97,94%	0,0309	0,9769	0,762s
REPTree	20,0%	94,23%	0,0423	0,9349	0,488s
VFI	20,0%	98,17%	0,0317	0,9795	2,285s
NNge	20,0%	98,13%	0,0316	0,979	8,917s
JRip	20,0%	97,9%	0,033	0,9764	9,282s
KStar	20,0%	98,61%	0,0254	0,9844	894,355s
Ridor	20,0%	96,88%	0,041	0,965	5,916s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	98,16%	0,0307	0,9794	2,49s
HyperPipes	30,0%	95,76%	0,1066	0,9521	0,388s
IBk	30,0%	98,83%	0,024	0,9868	27,582s
J48	30,0%	98,51%	0,0269	0,9833	0,352s
LibSVM	30,0%	95,07%	0,0515	0,9444	70,032s
NaiveBayes	30,0%	98,03%	0,0316	0,9779	4,434s
OneR	30,0%	94,33%	0,0554	0,9362	0,13s
PART	30,0%	98,2%	0,0288	0,9799	0,912s
REPTree	30,0%	95,47%	0,038	0,949	0,687s
VFI	30,0%	98,35%	0,0308	0,9815	2,049s
NNge	30,0%	98,47%	0,0287	0,9829	13,496s
JRip	30,0%	98,29%	0,0295	0,9808	8,851s
KStar	30,0%	98,79%	0,0236	0,9864	1189,522s
Ridor	30,0%	97,41%	0,0374	0,971	12,08s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	98,17%	0,0306	0,9795	2,414s
HyperPipes	40,0%	95,72%	0,1101	0,9518	0,377s
IBk	40,0%	98,89%	0,0229	0,9876	33,943s
J48	40,0%	98,52%	0,027	0,9834	0,421s
LibSVM	40,0%	96,09%	0,0459	0,9559	91,029s
NaiveBayes	40,0%	98,1%	0,0311	0,9787	4,183s
OneR	40,0%	94,89%	0,0526	0,9426	0,16s
PART	40,0%	98,39%	0,0275	0,9819	1,084s
REPTree	40,0%	96,11%	0,0357	0,9562	1,003s
VFI	40,0%	98,38%	0,0311	0,9819	1,775s
NNge	40,0%	98,52%	0,0283	0,9833	18,338s
JRip	40,0%	98,45%	0,0281	0,9826	9,675s
KStar	40,0%	98,84%	0,0228	0,987	1362,311s
Ridor	40,0%	97,73%	0,035	0,9745	16,949s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	98,23%	0,0301	0,9802	2,044s
HyperPipes	50,0%	95,3%	0,1146	0,9469	0,321s
IBk	50,0%	98,96%	0,0222	0,9883	37,604s
J48	50,0%	98,57%	0,0263	0,984	0,468s
LibSVM	50,0%	96,62%	0,0427	0,9619	98,63s
NaiveBayes	50,0%	98,12%	0,0311	0,9789	3,371s
OneR	50,0%	95,25%	0,0506	0,9467	0,117s
PART	50,0%	98,38%	0,0277	0,9818	1,179s
REPTree	50,0%	96,42%	0,0343	0,9598	1,301s
VFI	50,0%	98,52%	0,0299	0,9835	1,482s
NNge	50,0%	98,63%	0,0271	0,9847	23,528s
JRip	50,0%	98,53%	0,0271	0,9835	13,085s
KStar	50,0%	98,91%	0,0221	0,9877	1430,505s
Ridor	50,0%	98,0%	0,0329	0,9776	21,409s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	98,3%	0,0295	0,981	1,635s
HyperPipes	60,0%	95,27%	0,1149	0,9467	0,265s
IBk	60,0%	98,99%	0,0217	0,9886	33,034s
J48	60,0%	98,61%	0,0261	0,9844	0,473s
LibSVM	60,0%	96,83%	0,0414	0,9643	111,06s
NaiveBayes	60,0%	98,26%	0,0299	0,9805	2,814s
OneR	60,0%	95,56%	0,049	0,9502	0,11s
PART	60,0%	98,62%	0,0256	0,9846	1,467s
REPTree	60,0%	96,85%	0,0327	0,9645	1,458s
VFI	60,0%	98,47%	0,03	0,9829	1,207s
NNge	60,0%	98,81%	0,0253	0,9867	29,08s
JRip	60,0%	98,65%	0,026	0,9848	17,512s
KStar	60,0%	98,99%	0,0212	0,9886	1353,385s
Ridor	60,0%	98,04%	0,0325	0,978	27,052s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	98,3%	0,0297	0,9809	1,347s
HyperPipes	70,0%	95,05%	0,1159	0,9441	0,209s
IBk	70,0%	99,05%	0,0211	0,9893	29,181s
J48	70,0%	98,79%	0,0246	0,9864	0,477s
LibSVM	70,0%	96,94%	0,0406	0,9656	107,83s
NaiveBayes	70,0%	98,19%	0,0303	0,9797	1,923s
OneR	70,0%	95,73%	0,0481	0,952	0,077s
PART	70,0%	98,63%	0,0256	0,9847	1,397s
REPTree	70,0%	97,09%	0,0317	0,9673	1,567s
VFI	70,0%	98,64%	0,0298	0,9847	0,914s
NNge	70,0%	98,71%	0,0263	0,9855	34,51s
JRip	70,0%	98,68%	0,0258	0,9852	20,695s
KStar	70,0%	99,03%	0,0207	0,9891	1204,51s
Ridor	70,0%	98,18%	0,0313	0,9796	35,333s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	98,23%	0,0301	0,9802	0,836s
HyperPipes	80,0%	94,96%	0,1177	0,943	0,136s
IBk	80,0%	99,16%	0,0203	0,9905	21,219s
J48	80,0%	98,78%	0,0245	0,9863	0,501s
LibSVM	80,0%	96,93%	0,0407	0,9655	116,228s
NaiveBayes	80,0%	98,22%	0,0301	0,98	1,304s
OneR	80,0%	96,04%	0,0462	0,9557	0,078s
PART	80,0%	98,73%	0,0245	0,9857	1,671s
REPTree	80,0%	97,32%	0,0305	0,9699	1,76s
VFI	80,0%	98,59%	0,0301	0,9842	0,625s
NNge	80,0%	98,81%	0,0252	0,9867	40,23s
JRip	80,0%	98,64%	0,026	0,9847	25,951s
KStar	80,0%	99,04%	0,0205	0,9892	917,94s
Ridor	80,0%	98,32%	0,0301	0,9812	40,525s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	98,46%	0,0283	0,9828	0,486s
HyperPipes	90,0%	94,77%	0,12	0,9407	0,079s
IBk	90,0%	99,11%	0,02	0,99	10,988s
J48	90,0%	98,69%	0,0252	0,9852	0,503s
LibSVM	90,0%	96,89%	0,041	0,965	129,346s
NaiveBayes	90,0%	98,12%	0,031	0,979	0,674s
OneR	90,0%	96,35%	0,0443	0,959	0,073s
PART	90,0%	98,74%	0,0244	0,9858	1,73s
REPTree	90,0%	97,39%	0,0301	0,9707	1,906s
VFI	90,0%	98,59%	0,0302	0,9841	0,324s
NNge	90,0%	98,97%	0,0234	0,9884	45,799s
JRip	90,0%	98,64%	0,0253	0,9848	29,787s
KStar	90,0%	99,05%	0,0207	0,9893	511,074s
Ridor	90,0%	98,4%	0,0294	0,982	50,856s

B.4. Lernkruve für NLCI-Koprus ohne Wörter

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	95,53%	0,0437	0,9498	3,507s
HyperPipes	1,0%	94,76%	0,0784	0,9409	0,517s
IBk	1,0%	95,24%	0,0463	0,9464	2,549s
J48	1,0%	93,87%	0,0506	0,9308	0,266s
LibSVM	1,0%	66,56%	0,1344	0,6058	9,192s
NaiveBayes	1,0%	95,27%	0,0439	0,9468	6,257s
OneR	1,0%	92,51%	0,0635	0,9153	0,166s
PART	1,0%	90,34%	0,0604	0,8916	0,355s
REPTree	1,0%	93,03%	0,0541	0,9212	0,172s
VFI	1,0%	93,57%	0,0554	0,928	2,117s
NNge	1,0%	95,76%	0,0474	0,9523	0,846s
JRip	1,0%	88,56%	0,0741	0,8692	0,242s
KStar	1,0%	95,54%	0,0439	0,9497	50,506s
Ridor	1,0%	78,79%	0,1068	0,7619	0,233s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	96,44%	0,0397	0,96	3,15s
HyperPipes	2,0%	95,87%	0,0798	0,9535	0,473s
IBk	2,0%	96,5%	0,0407	0,9607	3,876s
J48	2,0%	95,89%	0,0428	0,9538	0,25s
LibSVM	2,0%	72,65%	0,1216	0,6831	11,185s
NaiveBayes	2,0%	96,64%	0,0391	0,9623	5,639s
OneR	2,0%	94,58%	0,0541	0,939	0,148s
PART	2,0%	94,36%	0,0489	0,9367	0,327s
REPTree	2,0%	94,65%	0,0485	0,9399	0,192s
VFI	2,0%	94,94%	0,0543	0,9433	2,093s
NNge	2,0%	96,78%	0,0414	0,9638	0,967s
JRip	2,0%	94,82%	0,0512	0,9416	0,292s
KStar	2,0%	96,98%	0,0377	0,9661	100,077s
Ridor	2,0%	87,13%	0,0832	0,8551	0,35s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	97,38%	0,0354	0,9706	3,045s
HyperPipes	5,0%	95,24%	0,0918	0,9463	0,479s
IBk	5,0%	97,73%	0,0332	0,9745	8,441s
J48	5,0%	97,18%	0,0366	0,9683	0,26s
LibSVM	5,0%	88,26%	0,0796	0,8664	18,65s
NaiveBayes	5,0%	97,34%	0,036	0,9701	6,033s
OneR	5,0%	95,49%	0,0494	0,9493	0,164s
PART	5,0%	96,58%	0,039	0,9616	0,417s
REPTree	5,0%	96,88%	0,0378	0,965	0,191s
VFI	5,0%	96,99%	0,052	0,9663	2,421s
NNge	5,0%	97,36%	0,0377	0,9704	1,674s
JRip	5,0%	96,52%	0,0424	0,9609	0,573s
KStar	5,0%	97,58%	0,0333	0,9728	243,024s
Ridor	5,0%	95,17%	0,051	0,9458	0,827s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	97,68%	0,0339	0,974	2,883s
HyperPipes	10,0%	94,18%	0,0993	0,9343	0,495s
IBk	10,0%	98,18%	0,0301	0,9796	17,108s
J48	10,0%	97,81%	0,0324	0,9754	0,272s
LibSVM	10,0%	93,28%	0,0601	0,924	24,964s
NaiveBayes	10,0%	97,54%	0,0351	0,9724	5,187s
OneR	10,0%	95,73%	0,048	0,952	0,146s
PART	10,0%	97,2%	0,0354	0,9686	0,442s
REPTree	10,0%	97,54%	0,034	0,9724	0,19s
VFI	10,0%	96,71%	0,0542	0,9631	2,214s
NNge	10,0%	97,64%	0,0354	0,9736	2,731s
JRip	10,0%	97,2%	0,0381	0,9685	1,053s
KStar	10,0%	97,91%	0,0307	0,9765	457,909s
Ridor	10,0%	96,05%	0,0462	0,9557	1,699s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	97,79%	0,0333	0,9752	2,578s
HyperPipes	20,0%	92,61%	0,1066	0,9163	0,424s
IBk	20,0%	98,49%	0,0272	0,983	29,499s
J48	20,0%	98,29%	0,0289	0,9808	0,285s
LibSVM	20,0%	96,73%	0,042	0,9632	35,202s
NaiveBayes	20,0%	97,79%	0,0334	0,9752	4,979s
OneR	20,0%	95,98%	0,0466	0,9549	0,15s
PART	20,0%	97,9%	0,0306	0,9764	0,609s
REPTree	20,0%	98,09%	0,0302	0,9786	0,257s
VFI	20,0%	97,4%	0,0542	0,9708	2,325s
NNge	20,0%	98,18%	0,0312	0,9796	5,631s
JRip	20,0%	97,97%	0,0322	0,9773	2,54s
KStar	20,0%	98,19%	0,0286	0,9797	809,751s
Ridor	20,0%	96,92%	0,0408	0,9655	3,957s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	97,88%	0,0326	0,9762	2,271s
HyperPipes	30,0%	91,02%	0,1081	0,8981	0,375s
IBk	30,0%	98,6%	0,026	0,9843	37,77s
J48	30,0%	98,51%	0,0268	0,9833	0,299s
LibSVM	30,0%	97,03%	0,0401	0,9666	42,123s
NaiveBayes	30,0%	97,79%	0,033	0,9752	4,063s
OneR	30,0%	95,92%	0,047	0,9541	0,128s
PART	30,0%	98,25%	0,0285	0,9804	0,65s
REPTree	30,0%	98,31%	0,0285	0,9811	0,288s
VFI	30,0%	97,67%	0,0534	0,9739	1,896s
NNge	30,0%	98,21%	0,0309	0,9799	7,579s
JRip	30,0%	98,19%	0,0305	0,9797	9,075s
KStar	30,0%	98,26%	0,0278	0,9804	1094,957s
Ridor	30,0%	97,39%	0,0376	0,9707	6,884s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	97,8%	0,0329	0,9754	1,961s
HyperPipes	40,0%	90,67%	0,1129	0,894	0,332s
IBk	40,0%	98,74%	0,0248	0,9859	41,337s
J48	40,0%	98,62%	0,026	0,9846	0,338s
LibSVM	40,0%	97,03%	0,0401	0,9666	51,484s
NaiveBayes	40,0%	97,84%	0,0329	0,9758	3,803s
OneR	40,0%	96,04%	0,0463	0,9555	0,126s
PART	40,0%	98,37%	0,0276	0,9818	0,833s
REPTree	40,0%	98,51%	0,0268	0,9833	0,388s
VFI	40,0%	97,36%	0,0549	0,9705	1,801s
NNge	40,0%	98,5%	0,0284	0,9831	10,855s
JRip	40,0%	98,34%	0,0288	0,9814	8,075s
KStar	40,0%	98,43%	0,0264	0,9824	1252,823s
Ridor	40,0%	97,68%	0,0354	0,974	10,03s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	97,93%	0,0322	0,9767	1,646s
HyperPipes	50,0%	90,28%	0,1152	0,8895	0,281s
IBk	50,0%	98,79%	0,0243	0,9864	44,078s
J48	50,0%	98,71%	0,0253	0,9856	0,321s
LibSVM	50,0%	96,99%	0,0403	0,9662	51,618s
NaiveBayes	50,0%	97,88%	0,0325	0,9763	2,909s
OneR	50,0%	96,15%	0,0456	0,9568	0,108s
PART	50,0%	98,43%	0,0271	0,9824	0,868s
REPTree	50,0%	98,6%	0,026	0,9843	0,345s
VFI	50,0%	97,64%	0,0541	0,9735	1,375s
NNge	50,0%	98,34%	0,0297	0,9813	11,803s
JRip	50,0%	98,47%	0,0279	0,9829	8,295s
KStar	50,0%	98,4%	0,0264	0,9821	1308,858s
Ridor	50,0%	97,92%	0,0335	0,9767	15,248s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	98,0%	0,0317	0,9775	1,335s
HyperPipes	60,0%	90,5%	0,1161	0,892	0,232s
IBk	60,0%	98,83%	0,0239	0,9869	40,532s
J48	60,0%	98,69%	0,0253	0,9853	0,334s
LibSVM	60,0%	97,17%	0,0391	0,9682	54,83s
NaiveBayes	60,0%	97,84%	0,0327	0,9757	2,333s
OneR	60,0%	96,16%	0,0456	0,9568	0,102s
PART	60,0%	98,63%	0,0256	0,9846	1,008s
REPTree	60,0%	98,65%	0,0255	0,9848	0,425s
VFI	60,0%	97,8%	0,055	0,9753	1,112s
NNge	60,0%	98,34%	0,0298	0,9814	15,241s
JRip	60,0%	98,47%	0,0277	0,9829	10,436s
KStar	60,0%	98,44%	0,0262	0,9825	1244,74s
Ridor	60,0%	97,98%	0,033	0,9773	19,481s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	97,91%	0,0322	0,9766	1,036s
HyperPipes	70,0%	89,89%	0,1186	0,8849	0,183s
IBk	70,0%	98,88%	0,0233	0,9874	37,661s
J48	70,0%	98,75%	0,0247	0,986	0,365s
LibSVM	70,0%	97,4%	0,0375	0,9708	56,837s
NaiveBayes	70,0%	97,88%	0,0322	0,9763	1,763s
OneR	70,0%	96,16%	0,0456	0,9568	0,08s
PART	70,0%	98,65%	0,0254	0,9849	1,068s
REPTree	70,0%	98,7%	0,0249	0,9855	0,507s
VFI	70,0%	97,82%	0,0551	0,9755	0,845s
NNge	70,0%	98,66%	0,0269	0,9849	17,639s
JRip	70,0%	98,64%	0,0261	0,9848	13,379s
KStar	70,0%	98,41%	0,0263	0,9822	1099,482s
Ridor	70,0%	98,13%	0,0317	0,979	22,605s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	98,06%	0,0314	0,9782	0,794s
HyperPipes	80,0%	89,62%	0,1211	0,8816	0,145s
IBk	80,0%	98,89%	0,0234	0,9875	30,029s
J48	80,0%	98,8%	0,0243	0,9866	0,414s
LibSVM	80,0%	97,45%	0,0371	0,9714	65,425s
NaiveBayes	80,0%	97,76%	0,0332	0,9749	1,328s
OneR	80,0%	96,15%	0,0456	0,9568	0,09s
PART	80,0%	98,68%	0,025	0,9852	1,268s
REPTree	80,0%	98,66%	0,0252	0,985	0,626s
VFI	80,0%	97,75%	0,0553	0,9748	0,571s
NNge	80,0%	98,33%	0,0296	0,9813	19,551s
JRip	80,0%	98,64%	0,0262	0,9848	16,982s
KStar	80,0%	98,63%	0,0246	0,9846	835,277s
Ridor	80,0%	98,04%	0,0325	0,978	28,919s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	98,03%	0,0313	0,9778	0,441s
HyperPipes	90,0%	89,35%	0,1211	0,8788	0,084s
IBk	90,0%	98,95%	0,0227	0,9882	16,562s
J48	90,0%	98,8%	0,0244	0,9865	0,365s
LibSVM	90,0%	97,36%	0,0377	0,9703	72,787s
NaiveBayes	90,0%	97,98%	0,031	0,9772	0,683s
OneR	90,0%	96,24%	0,0451	0,9577	0,075s
PART	90,0%	98,63%	0,0254	0,9847	1,461s
REPTree	90,0%	98,86%	0,0237	0,9872	0,717s
VFI	90,0%	97,69%	0,0558	0,9741	0,337s
NNge	90,0%	98,46%	0,0284	0,9827	25,323s
JRip	90,0%	98,6%	0,0265	0,9844	22,041s
KStar	90,0%	98,58%	0,0246	0,9841	465,937s
Ridor	90,0%	98,17%	0,0314	0,9795	33,802s

B.5. Lernkurve für PARSE-Korpus it Wörtern

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	73,69%	0,1223	0,6996	0,253s
HyperPipes	1,0%	73,28%	0,1376	0,6954	0,037s
IBk	1,0%	78,19%	0,1322	0,7472	0,072s
J48	1,0%	73,35%	0,1326	0,6906	0,02s
LibSVM	1,0%	21,31%	0,2673	0,0368	0,488s
NaiveBayes	1,0%	76,8%	0,1142	0,7347	0,451s
OneR	1,0%	50,14%	0,2118	0,4597	0,015s
PART	1,0%	59,06%	0,1509	0,5383	0,025s
REPTree	1,0%	44,4%	0,171	0,3278	0,016s
VFI	1,0%	73,65%	0,1329	0,7013	0,136s
NNge	1,0%	75,59%	0,1459	0,7201	0,058s
JRip	1,0%	48,47%	0,174	0,3812	0,023s
KStar	1,0%	78,42%	0,119	0,7511	0,795s
Ridor	1,0%	34,33%	0,2431	0,2561	0,018s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	85,5%	0,0953	0,8347	0,248s
HyperPipes	2,0%	85,91%	0,1187	0,8393	0,037s
IBk	2,0%	86,43%	0,1021	0,8441	0,105s
J48	2,0%	82,11%	0,1049	0,7931	0,02s
LibSVM	2,0%	20,89%	0,2679	0,0337	0,575s
NaiveBayes	2,0%	85,34%	0,0959	0,8321	0,457s
OneR	2,0%	63,1%	0,1828	0,5949	0,015s
PART	2,0%	73,89%	0,1235	0,7054	0,028s
REPTree	2,0%	64,92%	0,1337	0,5898	0,018s
VFI	2,0%	86,25%	0,0951	0,8433	0,154s
NNge	2,0%	84,43%	0,1171	0,8212	0,072s
JRip	2,0%	69,52%	0,1405	0,6382	0,031s
KStar	2,0%	87,02%	0,0961	0,8507	1,519s
Ridor	2,0%	53,37%	0,2052	0,4636	0,023s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	91,1%	0,0793	0,8986	0,241s
HyperPipes	5,0%	93,9%	0,1102	0,9307	0,037s
IBk	5,0%	95,82%	0,0599	0,9524	0,173s
J48	5,0%	90,02%	0,079	0,8862	0,022s
LibSVM	5,0%	51,17%	0,2103	0,404	0,838s
NaiveBayes	5,0%	92,47%	0,0748	0,9141	0,432s
OneR	5,0%	79,38%	0,1365	0,77	0,013s
PART	5,0%	88,93%	0,0885	0,8747	0,034s
REPTree	5,0%	79,53%	0,1009	0,7617	0,02s
VFI	5,0%	92,66%	0,0725	0,9169	0,164s
NNge	5,0%	94,71%	0,0687	0,9398	0,106s
JRip	5,0%	87,21%	0,0989	0,8526	0,047s
KStar	5,0%	93,91%	0,0665	0,9304	3,726s
Ridor	5,0%	78,15%	0,1407	0,7504	0,051s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	94,3%	0,066	0,9351	0,229s
HyperPipes	10,0%	96,08%	0,1162	0,9556	0,036s
IBk	10,0%	96,63%	0,0521	0,9617	0,27s
J48	10,0%	94,69%	0,0605	0,9396	0,022s
LibSVM	10,0%	63,93%	0,1809	0,5697	1,203s
NaiveBayes	10,0%	94,0%	0,0688	0,9316	0,407s
OneR	10,0%	86,81%	0,1095	0,8517	0,012s
PART	10,0%	93,95%	0,0654	0,9314	0,044s
REPTree	10,0%	87,79%	0,0792	0,8593	0,023s
VFI	10,0%	95,43%	0,0599	0,9483	0,174s
NNge	10,0%	96,31%	0,0574	0,9581	0,168s
JRip	10,0%	91,92%	0,0797	0,9076	0,073s
KStar	10,0%	96,88%	0,0487	0,9645	6,993s
Ridor	10,0%	90,0%	0,0945	0,8864	0,098s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	95,89%	0,0548	0,9534	0,206s
HyperPipes	20,0%	96,01%	0,1233	0,9547	0,033s
IBk	20,0%	97,46%	0,0455	0,9711	0,401s
J48	20,0%	97,04%	0,047	0,9665	0,023s
LibSVM	20,0%	78,9%	0,1383	0,7526	1,805s
NaiveBayes	20,0%	94,76%	0,064	0,9403	0,404s
OneR	20,0%	91,12%	0,0897	0,8998	0,013s
PART	20,0%	95,77%	0,0549	0,9521	0,061s
REPTree	20,0%	90,72%	0,0692	0,8936	0,032s
VFI	20,0%	96,79%	0,0506	0,9637	0,179s
NNge	20,0%	97,38%	0,0485	0,9702	0,325s
JRip	20,0%	94,96%	0,0656	0,9426	0,159s
KStar	20,0%	97,62%	0,0431	0,973	13,701s
Ridor	20,0%	93,4%	0,0768	0,9252	0,22s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	95,92%	0,0555	0,9536	0,202s
HyperPipes	30,0%	96,21%	0,1268	0,957	0,032s
IBk	30,0%	98,0%	0,0409	0,9773	0,538s
J48	30,0%	97,52%	0,0445	0,9718	0,028s
LibSVM	30,0%	82,19%	0,127	0,7926	2,249s
NaiveBayes	30,0%	95,55%	0,0587	0,9494	0,352s
OneR	30,0%	92,21%	0,084	0,9121	0,012s
PART	30,0%	96,68%	0,0494	0,9623	0,071s
REPTree	30,0%	92,43%	0,0637	0,9134	0,037s
VFI	30,0%	96,59%	0,0513	0,9614	0,167s
NNge	30,0%	97,9%	0,0432	0,9762	0,493s
JRip	30,0%	95,83%	0,0602	0,9526	0,259s
KStar	30,0%	98,09%	0,0387	0,9784	16,325s
Ridor	30,0%	94,55%	0,0702	0,9381	0,34s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	96,63%	0,051	0,9617	0,176s
HyperPipes	40,0%	95,94%	0,1286	0,9541	0,028s
IBk	40,0%	98,32%	0,0375	0,9809	0,576s
J48	40,0%	97,84%	0,0424	0,9755	0,028s
LibSVM	40,0%	88,63%	0,1011	0,8692	2,64s
NaiveBayes	40,0%	95,97%	0,0561	0,9542	0,299s
OneR	40,0%	93,03%	0,0794	0,9213	0,011s
PART	40,0%	97,18%	0,0465	0,968	0,093s
REPTree	40,0%	94,08%	0,0577	0,9324	0,041s
VFI	40,0%	96,76%	0,0516	0,9633	0,144s
NNge	40,0%	98,06%	0,0419	0,978	0,644s
JRip	40,0%	96,33%	0,0566	0,9582	0,33s
KStar	40,0%	98,43%	0,0358	0,9822	19,03s
Ridor	40,0%	95,22%	0,0656	0,9457	0,472s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	96,65%	0,051	0,962	0,135s
HyperPipes	50,0%	96,16%	0,1315	0,9565	0,022s
IBk	50,0%	98,56%	0,0344	0,9836	0,529s
J48	50,0%	98,05%	0,0399	0,9779	0,027s
LibSVM	50,0%	90,98%	0,09	0,8965	2,778s
NaiveBayes	50,0%	96,62%	0,0514	0,9617	0,231s
OneR	50,0%	93,8%	0,0748	0,9299	0,009s
PART	50,0%	97,18%	0,0467	0,9679	0,081s
REPTree	50,0%	94,9%	0,0532	0,9418	0,04s
VFI	50,0%	97,42%	0,0499	0,9708	0,111s
NNge	50,0%	98,33%	0,0389	0,981	0,748s
JRip	50,0%	97,14%	0,05	0,9674	0,401s
KStar	50,0%	98,33%	0,0372	0,9811	19,405s
Ridor	50,0%	95,77%	0,0619	0,952	0,572s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	97,06%	0,0467	0,9665	0,111s
HyperPipes	60,0%	95,24%	0,1343	0,9461	0,019s
IBk	60,0%	98,67%	0,0334	0,9849	0,499s
J48	60,0%	98,15%	0,0397	0,979	0,027s
LibSVM	60,0%	92,21%	0,084	0,9108	3,022s
NaiveBayes	60,0%	96,74%	0,0499	0,9629	0,183s
OneR	60,0%	94,57%	0,0701	0,9387	0,008s
PART	60,0%	97,76%	0,0425	0,9746	0,097s
REPTree	60,0%	95,59%	0,0516	0,9495	0,047s
VFI	60,0%	97,4%	0,0525	0,9705	0,092s
NNge	60,0%	98,36%	0,0382	0,9814	0,897s
JRip	60,0%	97,48%	0,0471	0,9714	0,503s
KStar	60,0%	98,66%	0,033	0,9848	20,444s
Ridor	60,0%	95,94%	0,0605	0,954	0,8s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	97,21%	0,045	0,9682	0,098s
HyperPipes	70,0%	94,54%	0,1356	0,9379	0,016s
IBk	70,0%	98,73%	0,0329	0,9855	0,481s
J48	70,0%	97,99%	0,0406	0,9772	0,032s
LibSVM	70,0%	92,58%	0,0821	0,9152	3,609s
NaiveBayes	70,0%	96,72%	0,0503	0,9626	0,139s
OneR	70,0%	95,49%	0,0635	0,949	0,008s
PART	70,0%	97,53%	0,044	0,9719	0,096s
REPTree	70,0%	95,54%	0,0523	0,9492	0,05s
VFI	70,0%	97,58%	0,0497	0,9725	0,071s
NNge	70,0%	98,48%	0,0369	0,9828	1,055s
JRip	70,0%	97,32%	0,0482	0,9694	0,577s
KStar	70,0%	98,59%	0,0339	0,984	16,331s
Ridor	70,0%	96,1%	0,0594	0,9558	0,99s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	96,74%	0,0496	0,9629	0,071s
HyperPipes	80,0%	94,21%	0,1362	0,9344	0,012s
IBk	80,0%	98,74%	0,0322	0,9857	0,362s
J48	80,0%	98,18%	0,0384	0,9794	0,033s
LibSVM	80,0%	93,03%	0,0793	0,9201	3,756s
NaiveBayes	80,0%	96,94%	0,046	0,9654	0,104s
OneR	80,0%	95,61%	0,0629	0,9503	0,008s
PART	80,0%	97,45%	0,0456	0,9709	0,117s
REPTree	80,0%	95,61%	0,0509	0,95	0,061s
VFI	80,0%	97,61%	0,0496	0,9728	0,053s
NNge	80,0%	98,25%	0,0387	0,9802	1,354s
JRip	80,0%	97,72%	0,0445	0,9741	0,808s
KStar	80,0%	98,41%	0,036	0,982	12,961s
Ridor	80,0%	96,28%	0,0578	0,9578	1,028s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	97,88%	0,0393	0,9758	0,041s
HyperPipes	90,0%	94,47%	0,1397	0,9374	0,007s
IBk	90,0%	98,57%	0,0326	0,9838	0,187s
J48	90,0%	97,97%	0,04	0,9769	0,031s
LibSVM	90,0%	92,95%	0,0791	0,9188	3,717s
NaiveBayes	90,0%	97,37%	0,0419	0,97	0,054s
OneR	90,0%	95,48%	0,0635	0,9487	0,007s
PART	90,0%	98,48%	0,0311	0,9827	0,129s
REPTree	90,0%	96,73%	0,0432	0,9625	0,066s
VFI	90,0%	97,79%	0,0502	0,9751	0,028s
NNge	90,0%	98,16%	0,0402	0,9791	1,542s
JRip	90,0%	97,88%	0,0424	0,9758	0,947s
KStar	90,0%	98,34%	0,0368	0,9811	7,143s
Ridor	90,0%	96,73%	0,0534	0,9627	1,176s

B.6. Lernkurve für PARSE-Korpus ohne Wörter

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	1,0%	78,87%	0,1097	0,7588	0,233s
HyperPipes	1,0%	80,05%	0,1263	0,7712	0,036s
IBk	1,0%	74,3%	0,1373	0,6998	0,072s
J48	1,0%	74,24%	0,1257	0,7033	0,021s
LibSVM	1,0%	21,73%	0,2665	0,05	0,44s
NaiveBayes	1,0%	74,48%	0,1221	0,7098	0,413s
OneR	1,0%	75,79%	0,1477	0,726	0,014s
PART	1,0%	59,87%	0,1523	0,5458	0,024s
REPTree	1,0%	73,24%	0,1255	0,6894	0,015s
VFI	1,0%	78,47%	0,1213	0,7526	0,129s
NNge	1,0%	78,18%	0,1401	0,7476	0,05s
JRip	1,0%	53,7%	0,1671	0,4399	0,02s
KStar	1,0%	77,18%	0,1235	0,7352	0,733s
Ridor	1,0%	43,25%	0,2258	0,3219	0,017s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	2,0%	85,03%	0,0979	0,8286	0,231s
HyperPipes	2,0%	87,3%	0,1138	0,8554	0,037s
IBk	2,0%	88,66%	0,0947	0,8698	0,097s
J48	2,0%	82,14%	0,1076	0,7944	0,021s
LibSVM	2,0%	31,72%	0,2481	0,1638	0,516s
NaiveBayes	2,0%	86,04%	0,0969	0,8398	0,408s
OneR	2,0%	82,05%	0,1273	0,7958	0,014s
PART	2,0%	74,73%	0,1256	0,7125	0,026s
REPTree	2,0%	83,78%	0,1046	0,8138	0,015s
VFI	2,0%	84,65%	0,1006	0,8255	0,139s
NNge	2,0%	86,92%	0,1081	0,8505	0,058s
JRip	2,0%	72,66%	0,1353	0,677	0,024s
KStar	2,0%	86,85%	0,0943	0,849	1,399s
Ridor	2,0%	57,67%	0,1956	0,5058	0,021s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	5,0%	91,45%	0,0788	0,9027	0,224s
HyperPipes	5,0%	93,04%	0,1125	0,9209	0,036s
IBk	5,0%	94,08%	0,0676	0,9324	0,179s
J48	5,0%	90,72%	0,079	0,8943	0,021s
LibSVM	5,0%	60,3%	0,1896	0,5217	0,741s
NaiveBayes	5,0%	91,28%	0,0807	0,9006	0,397s
OneR	5,0%	87,93%	0,1044	0,8628	0,013s
PART	5,0%	88,5%	0,0861	0,8695	0,033s
REPTree	5,0%	88,12%	0,0893	0,8641	0,016s
VFI	5,0%	90,75%	0,0784	0,8951	0,153s
NNge	5,0%	93,39%	0,0762	0,9248	0,084s
JRip	5,0%	85,69%	0,1045	0,8347	0,039s
KStar	5,0%	93,92%	0,0683	0,9307	3,435s
Ridor	5,0%	81,73%	0,1281	0,7913	0,042s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	10,0%	93,35%	0,0712	0,9243	0,212s
HyperPipes	10,0%	94,46%	0,1184	0,9373	0,036s
IBk	10,0%	96,88%	0,0503	0,9646	0,278s
J48	10,0%	94,45%	0,0632	0,9368	0,021s
LibSVM	10,0%	73,81%	0,1537	0,69	0,983s
NaiveBayes	10,0%	93,76%	0,0708	0,9289	0,379s
OneR	10,0%	89,19%	0,0989	0,8772	0,013s
PART	10,0%	94,13%	0,0664	0,9335	0,037s
REPTree	10,0%	92,96%	0,0727	0,9201	0,016s
VFI	10,0%	91,51%	0,08	0,9041	0,157s
NNge	10,0%	96,29%	0,0577	0,9579	0,128s
JRip	10,0%	92,23%	0,0787	0,9112	0,061s
KStar	10,0%	96,21%	0,0532	0,9569	6,41s
Ridor	10,0%	89,55%	0,0971	0,8812	0,078s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	20,0%	94,28%	0,0674	0,935	0,192s
HyperPipes	20,0%	87,86%	0,1264	0,8653	0,032s
IBk	20,0%	97,57%	0,0444	0,9724	0,436s
J48	20,0%	96,58%	0,0504	0,9612	0,021s
LibSVM	20,0%	84,96%	0,1166	0,8258	1,42s
NaiveBayes	20,0%	94,31%	0,0681	0,9353	0,335s
OneR	20,0%	91,21%	0,0893	0,9	0,011s
PART	20,0%	96,23%	0,0513	0,9573	0,049s
REPTree	20,0%	96,22%	0,0538	0,957	0,019s
VFI	20,0%	92,31%	0,0784	0,9129	0,15s
NNge	20,0%	97,54%	0,0469	0,9721	0,222s
JRip	20,0%	95,27%	0,0637	0,9461	0,12s
KStar	20,0%	97,14%	0,0459	0,9675	11,39s
Ridor	20,0%	93,93%	0,074	0,9312	0,165s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	30,0%	95,0%	0,0613	0,9431	0,168s
HyperPipes	30,0%	86,61%	0,1271	0,851	0,029s
IBk	30,0%	97,8%	0,0427	0,975	0,542s
J48	30,0%	97,52%	0,0437	0,9719	0,022s
LibSVM	30,0%	89,2%	0,0985	0,8757	1,718s
NaiveBayes	30,0%	94,3%	0,0684	0,9351	0,293s
OneR	30,0%	91,48%	0,0878	0,9031	0,011s
PART	30,0%	96,43%	0,0507	0,9596	0,054s
REPTree	30,0%	97,35%	0,0464	0,9699	0,021s
VFI	30,0%	92,44%	0,0779	0,9146	0,137s
NNge	30,0%	98,01%	0,0421	0,9774	0,316s
JRip	30,0%	96,04%	0,0585	0,955	0,178s
KStar	30,0%	97,99%	0,0407	0,9772	14,964s
Ridor	30,0%	94,53%	0,0704	0,938	0,258s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	40,0%	95,57%	0,0593	0,9497	0,147s
HyperPipes	40,0%	84,44%	0,1331	0,8273	0,025s
IBk	40,0%	98,1%	0,0394	0,9784	0,605s
J48	40,0%	97,85%	0,0432	0,9756	0,021s
LibSVM	40,0%	92,08%	0,0846	0,9092	2,014s
NaiveBayes	40,0%	95,02%	0,0619	0,9433	0,251s
OneR	40,0%	91,58%	0,0874	0,9041	0,009s
PART	40,0%	97,14%	0,0465	0,9676	0,059s
REPTree	40,0%	97,68%	0,0436	0,9737	0,024s
VFI	40,0%	92,69%	0,0777	0,9174	0,12s
NNge	40,0%	98,09%	0,0414	0,9783	0,422s
JRip	40,0%	96,47%	0,0552	0,9599	0,255s
KStar	40,0%	97,76%	0,0417	0,9746	17,109s
Ridor	40,0%	95,35%	0,0648	0,9472	0,363s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	50,0%	95,78%	0,0581	0,952	0,124s
HyperPipes	50,0%	79,44%	0,1347	0,7728	0,021s
IBk	50,0%	98,22%	0,0382	0,9797	0,618s
J48	50,0%	98,07%	0,0402	0,9781	0,022s
LibSVM	50,0%	92,37%	0,083	0,9126	2,158s
NaiveBayes	50,0%	95,14%	0,0614	0,9448	0,21s
OneR	50,0%	91,89%	0,0858	0,9076	0,008s
PART	50,0%	97,75%	0,042	0,9745	0,065s
REPTree	50,0%	97,64%	0,0443	0,9733	0,025s
VFI	50,0%	92,17%	0,0802	0,9115	0,103s
NNge	50,0%	98,13%	0,0408	0,9788	0,516s
JRip	50,0%	96,79%	0,0529	0,9634	0,328s
KStar	50,0%	98,1%	0,0384	0,9784	19,273s
Ridor	50,0%	95,76%	0,062	0,952	0,529s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	60,0%	95,75%	0,0579	0,9517	0,102s
HyperPipes	60,0%	79,33%	0,138	0,7721	0,018s
IBk	60,0%	98,55%	0,0348	0,9835	0,586s
J48	60,0%	98,43%	0,0365	0,9821	0,022s
LibSVM	60,0%	93,25%	0,0782	0,9225	2,31s
NaiveBayes	60,0%	95,26%	0,0608	0,9461	0,169s
OneR	60,0%	91,91%	0,0857	0,9079	0,008s
PART	60,0%	97,74%	0,0421	0,9743	0,071s
REPTree	60,0%	98,01%	0,0406	0,9774	0,031s
VFI	60,0%	92,29%	0,0817	0,9129	0,084s
NNge	60,0%	98,43%	0,0375	0,9821	0,627s
JRip	60,0%	97,43%	0,0473	0,9707	0,403s
KStar	60,0%	98,34%	0,037	0,9812	17,126s
Ridor	60,0%	95,93%	0,0604	0,9537	0,623s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	70,0%	96,32%	0,0545	0,9583	0,08s
HyperPipes	70,0%	79,54%	0,1392	0,7743	0,015s
IBk	70,0%	98,59%	0,0346	0,984	0,513s
J48	70,0%	98,22%	0,0386	0,9798	0,023s
LibSVM	70,0%	94,33%	0,0717	0,9351	2,456s
NaiveBayes	70,0%	95,11%	0,0617	0,9445	0,128s
OneR	70,0%	92,21%	0,0839	0,9112	0,008s
PART	70,0%	97,79%	0,0421	0,9749	0,076s
REPTree	70,0%	97,99%	0,0408	0,9772	0,033s
VFI	70,0%	92,32%	0,0824	0,9133	0,063s
NNge	70,0%	98,02%	0,0419	0,9776	0,735s
JRip	70,0%	97,48%	0,0472	0,9714	0,486s
KStar	70,0%	98,33%	0,0384	0,981	14,99s
Ridor	70,0%	96,24%	0,0582	0,9573	0,788s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	80,0%	96,09%	0,056	0,9555	0,059s
HyperPipes	80,0%	78,83%	0,1389	0,7668	0,01s
IBk	80,0%	98,53%	0,0352	0,9833	0,393s
J48	80,0%	98,48%	0,0345	0,9827	0,024s
LibSVM	80,0%	94,78%	0,0684	0,9405	2,546s
NaiveBayes	80,0%	96,07%	0,0559	0,9554	0,087s
OneR	80,0%	92,41%	0,0828	0,9139	0,007s
PART	80,0%	97,91%	0,04	0,9762	0,084s
REPTree	80,0%	97,72%	0,0434	0,9742	0,035s
VFI	80,0%	91,68%	0,0849	0,9062	0,044s
NNge	80,0%	98,53%	0,0363	0,9833	0,845s
JRip	80,0%	97,52%	0,0461	0,9717	0,596s
KStar	80,0%	98,25%	0,0368	0,9802	11,411s
Ridor	80,0%	96,44%	0,0565	0,9597	0,853s

Name	Anteil	Erwartungswert	Varianz	Kappa	Dauer
BayesNet	90,0%	95,62%	0,0589	0,9503	0,037s
HyperPipes	90,0%	78,39%	0,1404	0,7618	0,007s
IBk	90,0%	98,06%	0,0392	0,978	0,229s
J48	90,0%	98,62%	0,0344	0,9843	0,026s
LibSVM	90,0%	95,53%	0,063	0,9489	2,73s
NaiveBayes	90,0%	95,62%	0,0573	0,9502	0,046s
OneR	90,0%	93,04%	0,0789	0,9211	0,006s
PART	90,0%	97,7%	0,0421	0,9739	0,09s
REPTree	90,0%	97,88%	0,0413	0,976	0,039s
VFI	90,0%	91,98%	0,0835	0,9089	0,025s
NNge	90,0%	98,66%	0,0327	0,9847	0,968s
JRip	90,0%	98,02%	0,0405	0,9773	0,732s
KStar	90,0%	98,48%	0,0362	0,9827	6,687s
Ridor	90,0%	96,96%	0,0517	0,9656	1,147s

C. Ausschnitt eines von REPTree erstellten Entscheidungsbaumes auf dem NLCI-Korpus ohne Wörter

```
stanford2bidir = DT : DT
stanford2bidir = NN
|   openNLP-MaxEnt = DT : NN
|   openNLP-MaxEnt = NN
|   |   berkeley = NN
|   |   |   jitar = NNP : NNP
|   |   |   jitar = JJS : RB
|   |   |   jitar = [SONST] : NN
|   |   berkeley = JJ
|   |   |   stanford2left = JJ : JJ
|   |   |   stanford2left = FW : UH
|   |   |   stanford2left = [SONST] : NN
|   |   berkeley = NNP : NNP
|   |   berkeley = UH : UH
|   |   berkeley = VBP
|   |   |   openNLP-Perc = VB : VBZ
|   |   |   openNLP-Perc = [SONST] : VBP
|   |   berkeley = [SONST] : NN
|   openNLP-MaxEnt = RB : RB
|   openNLP-MaxEnt = NNP : NNP
|   openNLP-MaxEnt = VB : VB
|   openNLP-MaxEnt = PRP : PRP
|   openNLP-MaxEnt = VBP : VBP
|   openNLP-MaxEnt = [SONST] : NN
stanford2bidir = VBZ : VBZ
[...]
```