Top-down induction of classifier trees

Top-down induction of classifier trees
Master-Thesis von Sören Schmidt aus Jena
Mai 2015

TECHNISCHE UNIVERSITÄT DARMSTADT

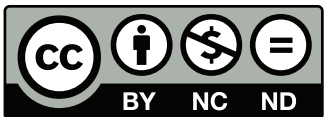
Fachbereich Informatik, Technische Universität Darmstadt
Fachgebiet Knowledge Engineering

Top-down induction of classifier trees
Top-down induction of classifier trees

Vorgelegte Master-Thesis von Sören Schmidt aus Jena

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten:

Tag der Einreichung:



Erklärung zur Master-Thesis

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

Darmstadt, den 26. Mai 2015

(S. Schmidt)

Zusammenfassung

Die Forschung in Multiklassen-Klassifikation ist darauf gerichtet, Klassifikatoren hinsichtlich ihrer Schnelligkeit und Genauigkeit zu optimieren. Dementsprechend werden immer wieder neue, effizientere Klassifikatoren vorgeschlagen. Der im Rahmen dieser Arbeit entwickelte Meta-Klassifikator Classifier Trees (CT) hat das Ziel, die Genauigkeit für bestehende Klassifikatoren zu erhöhen. CT erzeugt aus einem bestehenden Klassifikator einen Baum, indem ähnliche Klassen in den inneren Knoten zu Meta-Klassen zusammengefasst werden. Durch dieses Verfahren werden unterschiedliche Klassen früh im Baum voneinander getrennt, wodurch in tieferen Knoten ähnliche Klassen zusammengefasst sind. Die Erwartung ist, dass eine Abfolge von einfachen binären Entscheidungen ein genaueres Ergebnis liefert als eine Entscheidung zwischen vielen gleichrangigen Klassen.

In dieser Arbeit wird CT vorgestellt und seine Funktionsweise anhand eines Beispieles erläutert. Um das Ergebnis in der gewünschten Weise zu beeinflussen, wurden Parameter definiert, welche die Erzeugung der Meta-Klassen und die Struktur des Baumes bestimmen. Getestet wurde CT mit fünf verschiedenen Klassifikatoren auf siebzehn Datensätzen. Die Resultate zeigen, dass die Genauigkeit von CT unter Einsatz bestimmter Parameter für 55,3% der getesteten Datensatz-Klassifikator-Kombinationen höher ist als mit den ursprünglichen Klassifikatoren. Für vier der fünf Klassifikatoren zeigt CT auf vielen Datensätzen ein gutes Ergebnis mit einer teilweise signifikant verbesserten Genauigkeit gegenüber den ursprünglichen Klassifikatoren.

Inhaltsverzeichnis

1 Einführung **4**
- 1.1 Motivation . . . 4
- 1.2 Verwandte Arbeiten . . . 5

2 Grundbegriffe **8**
- 2.1 weka . . . 8
- 2.2 Beschreibung verwendeter Klassifikatoren . . . 8
  - 2.2.1 J48 / C4.5 . . . 8
  - 2.2.2 SMO . . . 9
  - 2.2.3 Naive Bayes . . . 9
  - 2.2.4 Random Forest . . . 10
  - 2.2.5 JRip / Ripper . . . 10

3 Classifier Trees **12**
- 3.1 Beschreibung des Algorithmus . . . 12
- 3.2 Ein kleines Beispiel . . . 14
- 3.3 Implementation . . . 17

4 Experimentelle Evaluierung **19**
- 4.1 Testspezifikationen . . . 19
- 4.2 Absolute Überlappung . . . 21
- 4.3 Relative Überlappung (-R) . . . 23
- 4.4 Absolute Balance (-aB) . . . 26
- 4.5 Balance mit Parameter (-B x) . . . 29
- 4.6 Baumstruktur . . . 32
- 4.7 Zusammenfassung . . . 33

5 Ausblick **38**

A Anhang **40**
- A.1 Visualisierungen der Unterschiede von CT zu den Ursprungsklassifikatoren . . . 40
  - A.1.1 Absolute Überlappung . . . 40
  - A.1.2 relative Überlappung . . . 43
  - A.1.3 absolute Balance . . . 46
  - A.1.4 Balance mit Parameter . . . 50
- A.2 Abbildungen der Baumstrukturen . . . 60

B Abkürzungsverzeichnis **63**

1 Einführung

1.1 Motivation

Multiklassen-Klassifikatoren können als Baum mit einer Wurzel plus einem Blatt pro Klasse interpretiert werden. Eine interessante Frage ist nun, ob es die Genauigkeit des Klassifikators verbessert, wenn dieser Baum in einen binären und eine schwere Entscheidung in eine Reihe von einfachen, welche das Ergebnis sukzessive verfeinern, gewandelt werden. Durch diese Aufteilung könnten Klassen, welche schwer unterscheidbar sind, in Abwesenheit von störenden anderen Klassen klassifiziert werden. Die vorliegende Arbeit verfolgt diese Idee und untersucht, ob die erwarteten Auswirkungen in der praktischen Anwendung erzielt werden können.

Bekannte Multiklassen-Klassifikatoren entscheiden oft direkt zwischen allen vorhandenen Klassen. Classifier Trees (CT) ist ein Meta-Klassifikator, der darauf abzielt, bestehende Multiklassen-Klassifikatoren zu verbessern, indem die oben genannten Ideen angewendet werden. Der Algorithmus erzeugt einen Baum, der aus einer Wurzel und Blattknoten für jede Klasse des Datensatzes besteht. In jeder Iteration werden für die Wurzel der Klassifikator gelernt, die zwei ähnlichsten Klassen anhand der Konfusionsmatrix ermittelt und in einer Meta-Klasse zusammengefasst, ein binärer Klassifikator für diese beiden gelernt, ein neuer Knoten für die Meta-Klasse an die Wurzel gehangen und die beiden Knoten der einzelnen Klassen wiederum an den Knoten der Meta-Klasse. Dies wird so lange wiederholt, bis der Baum binär ist. Ultimativ entsteht ein binärer Baum, bei dem sich in jedem nicht-Blatt-Knoten ein binärer Klassifikator befindet.

Abbildung 1 zeigt im linken Baum den Entscheidungsbaum eines normalen Multiklassen-Klassifikators. Der Klassifikator in der Wurzel entscheidet direkt über die Endausgabe, da alle Blätter an ihm hängen. Nach Anwendung von CT, abgebildet im rechten Baum, befindet sich in jedem nicht-Blatt ein Klassifikator. Dieser entscheidet jeweils, zu welchem Kindknoten weitergegangen wird. Erreicht man ein Blatt, so wird die Klasse zurückgegeben, welche dieses Blatt repräsentiert.

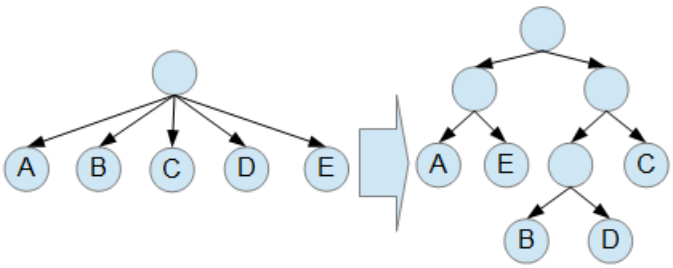


Abbildung 1: Baumstruktur vor und nach CT

Die Arbeit zeigt im nachfolgenden Abschnitt Verbindungen zu bereits existierenden Veröffentlichungen auf und schließt in Kapitel 2 mit einem Überblick über die theoretischen Grundlagen, wie zum Beispiel die benutzten Klassifikatoren, an. In Kapitel 3 werden der Algorithmus von CT, ergänzt durch ein kurzes Beispiel zum besseren Verständnis, und die Implementation beschrieben. Die Ergebnisse der praktischen Tests, deren Diskussion und Bewertung finden sich in

Kapitel 4. Mit einem Ausblick auf Ansätze zur weiteren Optimierung von CT schließt Kapitel 5 die Arbeit ab.

1.2 Verwandte Arbeiten

Im Folgenden werden solche Arbeiten aus der Forschung zur Multiklassen-Klassifikation vorgestellt und verglichen, welche das Konzept von CT in der vorliegenden Arbeit inspirierten oder ähnliche Ansätze verfolgen.

Ein erster möglicher Ansatz zur Erhöhung der Effizienz von Klassifikatoren besteht darin, Baumstrukturen zu erzeugen, um so die Klassifikation in mehrere Teilschritte zu zerlegen, wie ihn Bengio mit “Label Embedding Trees“ verfolgt [1]. Das Ziel dieses Vorgehens ist es, große Mengen an Klassen mit einem geringen Zeitaufwand zu bewältigen, ohne Genauigkeitsverlust im Vergleich zu anderen Klassifikatoren. Ein “label tree“ ist ein Baum, bei dem in jedem Knoten eine Menge von Labels und ein Label-Vorhersager, welcher anhand einer Instanz bestimmt zu welchem Kind weitergegangen wird, gespeichert sind. Ein Label entspricht einer Klasse und die Label-Menge eines Kindknotens ist immer eine Teilmenge des Vorgängerknotens. Die Anzahl der Blätter entspricht hier der Anzahl der Klassen (es gibt eine Variante von “label tree“, bei der es mehr Blätter als Klassen geben kann).

Der Algorithmus zum Lernen von “Label Embedding Trees“ besteht aus zwei Schritten: zuerst wird ein “label tree“ gelernt und anschließend findet auf diesem “label embedding“ statt. Dabei werden im ersten Schritt der Baum und die Label-Mengen ermittelt, indem für jede Klasse ein “one vs. rest“ Klassifikator gelernt und die Konfusionsmatrix erzeugt werden. Anhand dieser wird die Label-Menge in jedem Knoten auf die Kinder aufgeteilt. Es wird dafür ein Graph Schnitt Problem für jede Separation gelöst. Im zweiten Schritt werden die Label-Vorhersager ermittelt, welche durch die “label embeddings“ dargestellt werden. Diese “label embeddings“ werden als Klassifikator genutzt und mit einem spektralen Clustering aus der Konfusionsmatrix erzeugt. Für eine detaillierte Erklärung der Funktionsweise wird auf die Arbeit von Bengio verwiesen [1].

Aus Sicht der Klassifikation sind Bengios und der Ansatz der vorliegenden Arbeit identisch. Es wird in der Wurzel des Baumes gestartet, das Orakel (Klassifikator bzw. “label embeddings“) erhält die Instanz und gibt zurück, zu welchem Kind weitergegangen werden soll. Dies wird so lange wiederholt, bis ein Blatt erreicht wird. Der Unterschied zwischen beiden Ansätzen besteht in der Erzeugung der Klassifikatoren. Für “Label Embedding Trees“ werden der Baum und die Label-Mengen zuerst gelernt, die Orakel für die Knoten danach. Weiterhin wird der Baum anhand einer einzigen Konfusionsmatrix erzeugt. Für CT ändert sich die Konfusionsmatrix mit jeder Meta-Klasse (entspricht einer Label-Menge), die erzeugt wird. Weiterhin werden der Baum und die Klassifikatoren gleichzeitig gelernt. Beide Algorithmen ordnen ähnliche Klassen anhand der Konfusionsmatrix einer Label-Menge bzw. Meta-Klasse zu; wie diese Ähnlichkeit bestimmt wird, unterscheidet sich jedoch stark. CT fasst hintereinander immer zwei Klassen zusammen, “Label Embedding Trees“ teilt die Label immer auf die Kinder des aktuellen Knotens auf.

Einen ähnlichen Ansatz mit “label trees“ wählte Liu mit “Probabilistic Label Trees“ [7]. Hier werden die Parameter des “label trees“ mithilfe von Maximum-Likelihood-Schätzern ermittelt, was zu einer Steigerung der Genauigkeit führt. Der Algorithmus erzeugt nur die Label-Vorhersager,

ein Baum ist schon gegeben. Jeder Knoten s im Baum ist mit einer kategorischen Wahrscheinlichkeitsverteilung $p(y|S)$ verbunden, wobei y das Label der Testinstanz darstellt und S das Event, bei dem die Klassifikation den Knoten s erreicht. Um anschließend die Vorhersager zu erzeugen, wird eine rekursive Expansion der Knoten, startend bei der Wurzel, durchgeführt. Iterativ wird für jeden nicht-Blatt-Knoten zuerst ein Maximum-Likelihood-Klassifikator, basierend auf den kategorischen Wahrscheinlichkeitsverteilungen der einzelnen Kindknoten, gelernt und danach die kategorische Wahrscheinlichkeitsverteilung, die mit jedem Kindknoten verbunden ist.

Wie zuvor ist das Endergebnis aus Sicht der Klassifikation dasselbe. Wieder unterscheiden sich beide Ansätze in der Erzeugung des Klassifikators. Auch bei Liu werden der Baum und die Orakel in den Knoten hintereinander gelernt. Die Orakel bestehen aus Maximum-Likelihood-Klassifikatoren gegenüber dem gegebenen bei CT. Letztlich werden die Orakel rekursiv, startend bei der Wurzel, erzeugt, während CT in jeder Iteration auf der Wurzel arbeitet.

Ein anderer Ansatz als “label trees“ wird in den nächsten zwei Arbeiten verfolgt. In der ersten schlägt Platt den Algorithmus DAGSVM vor [9]. Dieser ermöglicht es Multiklassen Support-Vektor-Maschinen (SVM) schneller als herkömmlich zu trainieren und zu evaluieren. Die dabei entstehende Struktur ist ein “Decision Directed Acyclic Graph“, kurz DDAG.

Wie der Name bereits sagt, ist ein DDAG ein azyklischer gerichteter Graph, bei dem sich in jedem Knoten ein Klassifikator befindet. Es gibt genau einen Knoten, genannt Wurzel, der keinen Vorgänger besitzt und jeder Knoten hat zwei Nachfolger oder ist ein Blatt. Die Knoten sind wie folgt in einer Pyramide angeordnet: es gibt n Ebenen, die Anzahl der Knoten auf einer Ebene entspricht der Nummer der Ebene. Jeder Knoten (außer Blättern) hat zwei Verweise auf Knoten der nächsten Ebene, sodass die beiden Knoten an den Rändern einer Ebene der Pyramide einen Vorgänger besitzen, alle anderen zwei. Ein n Klassen Problem hat somit $n(n-1)/2$ nicht-Blatt-Knoten. Die Klassifikation startet in der Wurzel und geht zu dem Knoten weiter, den der Klassifikator mit der gegebenen Instanz vorhersagt. Die Klassifikatoren in einem DDAG sind dabei so aufgebaut, dass eine Entscheidung von einem solchen jeweils eine von zwei Klassen eliminiert. Eine Instanz startet also mit einer Liste aller Klassen und mit jedem Klassifikator wird eine davon entfernt. Wird ein Blatt erreicht, befindet sich nur noch eine Klasse in der Liste, welche zurückgegeben wird.

Der Algorithmus DAGSVM verwendet für die Klassifikatoren binäre SVM. Diese SVM werden jeweils nur aus den Instanzen des Trainingsdatensatzes gelernt, welche einer der beiden Klassen angehören, zwischen denen entschieden wird. Es handelt sich somit um einen “one vs. one“ Ansatz. Dadurch wird erreicht, dass die einzelnen Entscheidungen einfacher sind als bei einem “one vs. rest“ Ansatz.

Im Gegensatz zu den Arbeiten von Bengio und Liu ist das Endergebnis hier nicht identisch, da es sich bei einem DDAG nicht um einen Baum handelt. Auf unterschiedlichen Pfaden durch den Graphen kann dasselbe Blatt erreicht werden. Ähnlich zum Konzept von CT ist, dass jeder Klassifikator nur eine einfache Entscheidung treffen muss. Jedoch wird bei DAGSVM bei jeder Entscheidung nur eine Klasse eliminiert, bei CT eine Meta-Klasse. Bei der Klassifikation einer Instanz mit DAGSVM muss damit immer die gleiche Anzahl von Klassifikatoren durchlaufen werden, bei CT sind es maximal so viele wie bei einer DAGSVM, wenn beide die gleiche Anzahl von Klassen besitzen. Ähnlich ist weiterhin, dass die Menge der Instanzen, welche zum Lernen

der einzelnen Klassifikatoren benutzt wird, immer nur diejenigen Elemente enthält, welche den zur Wahl stehenden Klassen angehören. Allerdings sind dies bei CT häufig Meta-Klassen und nicht immer die ursprünglichen wie bei einer DAGSVM.

Das Konzept von Vural [11] verfolgt mit “divide-by-2“ (DB2) ebenfalls das Ziel, SVM für Multiklassen-Probleme schneller zu trainieren und zu evaluieren. DB2 erzeugt einen binären Baum, welcher $(n-1)$ nicht-Blatt-Knoten besitzt, wobei n die Anzahl der Klassen beschreibt. In jedem dieser Knoten befindet sich eine binäre SVM. Startend in der Wurzel wird die Trainingsdatenmenge in zwei Subsets geteilt mittels “k-means based division“, “Spherical shells“ oder in der Art und Weise, dass die Differenz der Größe beider Subsets minimal ist[1]. Für diese beiden Subsets wird anschließend eine SVM gelernt. Dieser Prozess setzt sich rekursiv auf den einzelnen Subsets fort, bis diese jeweils nur noch Instanzen einer einzigen Klasse beinhalten.

DB2 und CT weisen wesentliche Ähnlichkeiten, aber auch Unterschiede auf. Wie bei Bengio und Liu ist die Klassifikation identisch, indem in der Wurzel gestartet wird und der Klassifikator anhand der Testinstanz entscheidet, zu welchem Kind weitergegangen wird. Ebenfalls ist ähnlich, dass die Entscheidung zwischen zwei Subsets / Meta-Klassen statt findet. Die Algorithmen unterscheiden sich hauptsächlich darin, dass bei DB2 der Trainingsdatensatz in jeder Rekursion geteilt wird, während bei CT in jeder Iteration eine Klasse entfernt wird. Auch die verwendeten Methoden unterscheiden sich stark. Bei DB2 wird die Menge zuerst geteilt und anschließend der Klassifikator gelernt, während bei CT zuerst der Klassifikator gelernt und anhand des gelieferten Ergebnisses über das weitere Verfahren entschieden wird.

[1] Genaue Beschreibungen dieser Verfahren finden sich in [11]

2 Grundbegriffe

2.1 weka

weka steht für “Waikato Environment for Knowledge Analysis“ und ist ein Tool für maschinelles Lernen der Universität Waikato in Neuseeland [6]. Es wurde in Java entwickelt und ist frei verfügbar unter der GNU General Public License[2].

weka enthält viele Features für Datamining-Aufgaben. Es ermöglicht das Vorverarbeiten von Datensätzen, zum Beispiel Diskretisierung oder das Löschen von Attributen oder Klassen, und stellt eine Vielzahl von Klassifikatoren zur Verfügung. Weitere Funktionalitäten sind Clustering, Regression, das Finden von Assoziationsregeln und ROC-Kurven-Analyse. Datensätze und Ergebnisse können zum besseren Verständnis visualisiert werden.

Ebenso kann weka verwendet werden, um eigene Klassifikatoren zu implementieren. Dazu wird die .jar von weka in ein Java-Projekt eingefügt, womit die Funktionalitäten von weka in diesem nutzbar werden. So bietet weka Schablonen für alle möglichen Typen von Klassifikatoren, was eine eigene Implementation vereinfacht: im Wesentlichen müssen nur der Algorithmus zur Erzeugung des Klassifikators und zur Klassifikation einer Instanz implementiert werden. Ein Nachteil von weka besteht darin, dass die Manipulation von Datensätzen oder Instanzen umständlich ist.

2.2 Beschreibung verwendeter Klassifikatoren

In diesem Abschnitt werden die Herkunft und Funktionsweise der verwendeten Klassifikatoren vorgestellt. Für detaillierte Informationen wird auf die jeweils zitierten Veröffentlichungen verwiesen.

2.2.1 J48 / C4.5

C4.5, in weka J48 genannt, ist ein von Ross Quinlan [10] entwickelter Algorithmus zur Erzeugung von Entscheidungsbäumen und stellt eine Weiterentwicklung des ebenfalls von ihm stammenden ID3 Algorithmus dar.

ID3 erzeugt einen Entscheidungsbaum, indem er über die Attribute des Trainingsdatensatzes iteriert. Zunächst wird für jedes Attribut die Entropie oder der Information-Gain berechnet. Anschließend wird das Attribut ausgewählt, welches die kleinste Entropie oder den größten Information-Gain besitzt. Anhand dieses Attributes wird der Datensatz in so viele Mengen geteilt, wie es Belegungen für dieses Attribut gibt, womit die entsprechenden Instanzen jeweils ihren Mengen zugeordnet werden. Auf jeder dieser Mengen wird der Algorithmus rekursiv aufgerufen. Wenn eine Menge nur noch Elemente einer einzigen Klasse enthält und damit pur ist, handelt es sich um ein Blatt des Baumes und der Algorithmus bricht an dieser Stelle ab. Weiterhin wird bei jedem Aufruf das zur Trennung benutzte Attribut entfernt.

[2] http://www.gnu.org/licenses/gpl.html; zuletzt aufgerufen am 25.05.2015

Infolge seiner vielen Schwächen ist ID3 für eine Praxisanwendung ungeeignet. C4.5 erweitert diesen Algorithmus, indem kontinuierliche bzw. numerische Attribute passend diskretisiert werden. Weiterhin stellen Instanzen mit fehlenden Attribut-Werten kein Problem mehr dar, da diese bei der Erzeugung des Entscheidungsbaumes vernachlässigt werden. Wenn eine Instanz bei der Klassifikation an einen Knoten gelangt, der über ein Attribut entscheidet, das diese nicht besitzt, werden alle Pfade verfolgt und am Ende der Pfad mit der höchsten Popularität gewählt. Die dritte Erweiterung ist “Pruning“. Hierbei werden am erzeugten Entscheidungsbaum Äste abgeschnitten, wenn der geschätzte Fehler an einem Knoten größer ist, als in dem Unterbaum, den dieser Knoten induziert, was “Overfitting“ verhindert. Als letzte Erweiterung in C4.5 können Attribute gewichtet werden, sodass Entscheidungen anhand höher gewichteter Attribute möglichst tief im Baum stattfinden.

C4.5 ist schnell und erzielt häufig sehr gute Ergebnisse und wird dadurch nicht selten in der Praxis eingesetzt. Es existiert eine kommerzielle Variante C5.0, welche stärker ist als C4.5.

2.2.2 SMO

SMO ist eine Variation einer “Support vector machine“ (SVM), erfunden von John Platt [8]. SMO steht für “Sequential Minimal Optimization“ und ist ein Algorithmus, um eine SVM schneller zu trainieren.

Eine SVM ist ein binärer Klassifikator und betrachtet eine Instanz als einen n-dimensionalen Vektor, wobei n für die Anzahl der nicht-Klassen-Attribute steht. Eine lineare SVM platziert eine n-1 dimensionale Hyperebene in diesem n dimensionalen Raum, um die Punkte beider Klassen bestmöglich voneinander zu trennen. Dabei ist die Distanz zu dem der Hyperebene am nächsten gelegenen Punkt der beiden Klassen zu maximieren.

Bei nicht-linearen SVM wird eine Kernel-Funktion benutzt, um die Punkte in einen höher dimensionalen Raum zu transformieren. In diesem wird wiederum eine Hyperebene nach gegebenen Parametern berechnet und anschließend zurück transformiert. Es gibt verschiedene Kernel-Funktionen, die sich bewährt haben. Die Effektivität einer SVM hängt stark von der Wahl des Kernels und der Parameter ab.

Um Multiklassen-Probleme mit einer SVM lösen zu können, muss man diese in binäre Probleme zerlegen, zum Beispiel mit einer “one vs. one“ Zerlegung, und anschließend eine Mehrheitswahl vornehmen.

Die Berechnung einer SVM ist besonders für komplexe Datensätze sehr ressourcenintensiv. SMO zerlegt das Optimierungsproblem in mehrere kleinere und löst diese analytisch. Dadurch werden sowohl Zeit als auch Speicher gespart, was SVM praxistauglicher macht.

2.2.3 Naive Bayes

Naive Bayes (NB) ist eine Familie von einfachen probabilistischen Klassifikatoren, welche auf dem Satz von Bayes beruhen. Es wird angenommen, dass alle Attribute komplett unabhängig voneinander sind, was in der Praxis normalerweise nicht zutrifft.

Bei Naive Bayes ist eine Instanz durch ihre spezifischen Werte in den einzelnen Attributen $(x_1, ..., x_n)$ definiert. Um eine Instanz zu klassifizieren, muss für jede der k möglichen Klassen C eine Wahrscheinlichkeit $p(C_k|x_1, ..., x_n)$ bestimmt werden. Nach dem Satz von Bayes kann dies umformuliert werden zu $p(C_k|x) = p(C_k) * p(x|C_k)/p(x)$. In diesem Kontext stellt $p(x)$ effektiv eine Konstante dar und kann somit vernachlässigt werden. Mit der Kettenregel für bedingte Wahrscheinlichkeiten wird die Formel umgeschrieben zu $p(C_k) * p(x_1|C_k) * p(x_2|C_k, x_1) * ... * p(x_n|C_k, x_1, ..., x_{n-1})$. Mit Anwendung der Annahme, dass alle Attribute unabhängig voneinander sind, vereinfacht sich die Formel zu $p(C_k) * \prod_{i=1}^{n} p(x_i|C_k)$. Im Normalfall wird anschließend die Klasse ausgewählt, welche die höchste Wahrscheinlichkeit besitzt. Die zur Berechnung benutzten Grundwahrscheinlichkeiten werden bei Generierung des spezifischen Klassifikators aus dem Trainingsdatensatz ermittelt.

Trotz seiner Einfachheit produziert Naive Bayes gute Ergebnisse auch in komplexen realen Situationen. Durch ihre hohe Anpassbar- und Skalierbarkeit sind die Klassifikatoren der Familie sehr schnell lernbar. Allerdings existieren andere Klassifikatoren, welche Naive Bayes leistungsmäßig übertreffen [3]. Differenziertere Beschreibungen zur Familie von Naive Bayes Klassifikatoren finden sich zum Beispiel in der Arbeit von George H. John und Pat Langley [5].

2.2.4 Random Forest

Random Forest (RF) als Ensemble-Verfahren erzeugt nicht nur einen, sondern eine Vielzahl von Klassifikatoren. Um eine Instanz zu klassifizieren, wird diese von jedem Klassifikator des Ensembles klassifiziert, anschließend entscheidet eine Mehrheitswahl über das Endergebnis.

In dem von Leo Breiman stammenden Algorithmus für Random Forest [2] wird eine Vielzahl von Entscheidungsbäumen konstruiert. Für jeden der Bäume wird eine “Bootstrap“ Probe der Trainingsmenge erzeugt, aus welchen die einzelnen Bäume konstruiert werden. Eine “Bootstrap“ Probe wird gebildet, indem man n Elemente mit Zurücklegen aus der Trainingsmenge zieht, wobei n die Größe dieser Trainingsmenge darstellt. Ein Baum wird analog zu ID3 konstruiert mit dem Unterschied, dass die Wahl des Attributes, auf dem gesplittet wird, nicht aus allen verfügbaren erfolgt, sondern nur aus einer zufällig k großen Auswahl. k ist ein Parameter, der kleiner sein muss als die Anzahl der Attribute. Diese zufällige Auswahl wird an jedem Knoten neu bestimmt. Es findet kein anschließendes “Pruning“ statt wie bei C4.5. Die Klassifikation funktioniert analog zu anderen Ensemble-Verfahren, wie oben beschrieben.

Random Forest beinhaltet die positiven Eigenschaften anderer Entscheidungsbaum-Klassifikatoren, beseitigt aber das Problem des “Overfitting“. Weiterhin kann der Algorithmus stark parallelisiert werden, da alle Bäume unabhängig voneinander berechnet werden können, was zu einer erheblichen Beschleunigung führt.

2.2.5 JRip / Ripper

Ripper ist ein von William W. Cohen entwickelter propositionaler Regellerner [4] und eine Weiterentwicklung des Regellerners IREP. Ähnlich zu Entscheidungsbäumen werden die gelernten Regeln anschließend beschnitten, um überflüssige Bedingungen oder Regeln zu entfernen.

Konkret wird dabei über die Klassen des Trainingsdatensatzes von der am wenigsten vorkommenden Klasse zur häufigsten iteriert. Jede Iteration enthält drei Phasen.

Die erste Phase besteht aus zwei Schritten: im ersten wird eine Regel “gewachsen“, indem der Wert des Attributes mit dem höchsten Information-Gain ausgewählt und als Bedingung angehängt wird. Im zweiten Schritt werden die erzeugten Regeln inkrementell beschnitten. Diese beiden Schritte werden so lange wiederholt, bis die Beschreibungslänge des Regelsets und der Trainingsinstanzen 64 Bit größer ist als die kleinste bisher gefundene, es keine positiven Trainingsinstanzen mehr gibt oder die Fehlerrate größer ist als 50%.

In der zweiten Phase findet eine Optimierung des gefundenen Regelsets statt. Dazu werden für jede Regel aus zufälligen Daten zwei Varianten erzeugt, wobei die zufälligen Daten Ergebnis einer Wiederholung der Prozedur aus Phase eins sind. Anschließend wird die Variante mit der kürzesten Beschreibungslänge als Repräsentant für das Regelset ausgewählt. Wenn nach Optimierung aller Regeln noch nicht abgedeckte positive Trainingsinstanzen existieren, wird für jene Phase eins noch einmal aufgerufen.

In der dritten Phase werden solche Regeln entfernt, welche die Beschreibungslänge des finalen Regelsets bei Verbleib in diesem vergrößern würden. Das entstandene Regelset wird in das finale eingefügt, womit eine Iteration des Algorithmus abgeschlossen ist.

3 Classifier Trees

3.1 Beschreibung des Algorithmus

Classifier Trees (CT) ist ein Meta-Klassifikator für Multiklassen-Klassifizierung, der darauf abzielt, die Genauigkeit bestehender Klassifikatoren zu verbessern. Die Idee ist, den flachen Entscheidungsbaum in einen tieferen und binären zu wandeln, sodass an jedem Knoten nur eine einfache Entscheidung getroffen werden muss. Statt einer großen Entscheidung beim ursprünglichen Klassifikator wird das Ergebnis bei CT sukzessive verfeinert. Weiterhin wird angestrebt, dass ähnliche Klassen besser getrennt werden. Es ist üblicherweise einfacher zwei Klassen zu trennen, wenn nur diese beiden zur Auswahl stehen und andere nicht mit auf die Entscheidung einwirken. Durch die Umsetzung dieser Prinzipien soll die Anwendung von CT einen Genauigkeitsgewinn gegenüber dem ursprünglichen Klassifikator bewirken. In Abbildung 2 ist der Pseudocode einer Iteration von CT zu sehen.

```
1: Lerne Klassifikator für Datensatz und speichere ihn im Wurzelknoten
2: Finde zu fusionierende Klassen x und y mithilfe der Konfusionsmatrix
3: Wenn keine Klassen gefunden wurden, dann brich den Algorithmus ab
4: Erzeuge Datensatz z, welcher ausschließlich Instanzen der Klassen x
   und y enthält
5: Setze im Datensatz alle Instanzen der Klassen x oder y auf Klasse xy
6: Erzeuge Knoten xy und hänge diesen an die Wurzel
7: Entferne Knoten x und y von der Wurzel und hänge sie an Knoten xy
8: Lerne Klassifikator für Datensatz z und speichere ihn in Knoten xy
9: Wenn exakt zwei Knoten an der Wurzeln hängen, dann lerne Klassifikator
   für den Datensatz und speichere ihn in der Wurzel
```

Abbildung 2: Eine Iteration von CT in Pseudocode

Der Algorithmus arbeitet iterativ. In jeder Iteration werden zwei Knoten, welche an der Wurzel den Baumes hängen, durch einen Meta-Knoten ersetzt, an den wiederum die zwei ersetzten Knoten gehangen werden. Um dies zu erreichen, wird zuerst der vorgegebene Klassifikator auf dem Datensatz gelernt. Anhand der Konfusionsmatrix werden anschließend zwei Klassen ausgewählt, welche in einem Meta-Knoten zusammen gefasst werden sollen. Welche zwei Klassen ausgewählt werden, hängt zusätzlich zu den Werten der Konfusionsmatrix von den Eingabeparametern ab. Welche es gibt und wie sie die Wahl beeinflussen, wird an späterer Stelle erläutert.

Es werden zwei neue Datensätze erzeugt. Der erste entspricht dem alten Datensatz mit dem Unterschied, dass bei denjenigen Instanzen, welche zu einer der beiden zusammengefügten Klassen gehören, die Klasse auf die neu erzeugte Meta-Klasse gesetzt wird. Dieser Datensatz wird dann in der nächsten Iteration als Startdatensatz benutzt. Ein zweiter Datensatz besteht ausschließlich aus den Instanzen, welche einer der beiden zusammengefügten Klassen angehören. Auf diesem Datensatz wird der Klassifikator für den Meta-Knoten gelernt.

Die Anzahl der Iterationen beträgt maximal die Anzahl der Klassen im Datensatz minus zwei. Danach ist der Baum binär und der Algorithmus ist beendet. Er kann allerdings auch vorzeitig

abbrechen. Dies passiert, wenn in den betrachteten Kombinationen von Klassen in einer Iteration keine “false positives“ und “false negatives“ auftreten, der Klassifikator also perfekt die Instanzen ihren Klassen zuordnet.

Falls der Algorithmus nicht vorzeitig abgebrochen wurde, muss am Ende der letzten Iteration nochmals ein Klassifikator für die Wurzel gelernt werden, weil der zu Beginn der Iteration erzeugte zwischen drei Klassen entscheidet und nicht mehr aktuell ist. Ist der Klassifikator für die zwei an der Wurzel hängenden Klassen gelernt, dann ist der Algorithmus beendet und der Klassifikator für CT gelernt.

Um die zwei Klassen zu bestimmen, welche fusioniert werden sollen, werden sämtliche Paare von Klassen betrachtet. In der Standardvariante, im Folgenden auch -A für Absolut genannt, werden die beiden Klassen zusammengefasst, bei welchen die Summe der “false positives“ und “false negatives“, nachfolgend auch absolute Überlappung genannt, den absolut höchsten Wert beträgt. Die auf Grund dieser Vorgehensweise erwartete Baumstruktur besteht aus mehreren Ketten, welche an der Wurzel hängen, und ist sehr unbalanciert.

Mit einigen Parametern kann die Auswahl beeinflusst werden. Es wurden folgende Parameter festgelegt, die dies ermöglichen:

- **-R:** -R steht für relativ und bewirkt, dass nicht mehr die beiden Klassen mit der absolut höchsten Überlappung, sondern die mit der relativ niedrigsten Überlappung ausgewählt werden. Dafür wird die Summe der richtig klassifizierten Beispiele durch die Summe der falsch klassifizierten geteilt. Somit werden Klassen zusammengefasst, die relativ zu den richtig klassifizierten Instanzen viele falsch klassifizierte besitzen. Dies sollte zu weniger Ketten als bei -A führen, jedoch wird angenommen, dass der Baum dennoch sehr unbalanciert ist.
- **-aB:** -aB steht für absolute Balance und erzwingt, dass der Baum so balanciert wie möglich ist. Dafür werden bei der Auswahl der zwei Klassen nur diejenigen Paare betrachtet, bei deren Fusionierung der Baum ausgeglichen bleibt. Hierfür wird für jeden an der Wurzel hängenden Knoten die Tiefe des Subbaumes, den dieser induziert, berechnet. Nur Knoten mit minimaler Tiefe werden betrachtet. Wenn nur ein Knoten den kleinsten Wert besitzt, so muss dieser mit einem der verbleibenden zusammengefasst werden. Diese Option führt dazu, dass der Algorithmus oft vorzeitig abbricht, da wesentlich weniger Paare betrachtet werden.
- **-B x:** -B steht für Balance, wobei x einen Wert angibt, wie stark die Balance des Baumes bei der Auswahl der Klassen einfließen soll. Ein größerer Wert bedeutet, dass stärker auf eine ausgeglichene Balance geachtet wird. Hierfür wird wie bei -aB für jedes Kind der Wurzel die Tiefe des Subbaumes, den dieses Kind induziert, berechnet. Anschließend wird der ursprüngliche Wert, welcher über die Wahl entscheidet, multipliziert mit der Summe der Tiefe der beiden betrachteten Knoten, potenziert mit x. Wenn x Null ist, entspricht es der Grundversion. Wird x auf unendlich gesetzt, entspricht dies nicht -aB. In -B werden immer alle Knotenpaare betrachtet, bei -aB nur eine Auswahl. Dies führt zu unterschiedlichen Baumstrukturen, in Abhängigkeit von x.

Weiterhin kann der Parameter -R mit -aB oder -B kombiniert werden. Dann fließen bei der Auswahl nicht nur die Summe der falsch klassifizierten Instanzen in die Wahl ein, sondern

auch die richtigen. Die Baumstrukturen unterscheiden sich jedoch wenig im Gegensatz zu ohne -R, da -aB diese festschreibt und -B je nach Parameter die Balance beeinflusst. Wenn in den folgenden Betrachtungen -A, -aB oder -B x als Parameter gegeben sind, so bedeutet dies, dass zur Auswahl der Klassen absolute Überlappung benutzt wurde. Für -aB und -B x wird -A nicht zusätzlich hinzu geschrieben, da es sich nicht um einen Parameter für den Algorithmus handelt. Findet sich -R in der Parameterliste, so handelt es sich immer um relative Überlappung.

3.2 Ein kleines Beispiel

In diesem Abschnitt wird anhand eines einfachen fiktiven Datensatzes ein Klassifikator gelernt, um die Prozedur der Erzeugung eines CT-Klassifikators besser zu verdeutlichen. Es wird die Standardkonfiguration von CT verwendet, bei welcher die zwei Knoten ausgewählt werden, welche die absolut höchste Überlappung aufweisen. Weiterhin wird nicht darauf geachtet, ob der Baum balanciert ist. In der nachfolgenden Tabelle ist die Verteilung der Instanzen auf die Klassen zu sehen.

Klasse	A	B	C	D	E
Anzahl	26	30	29	32	28

Es wird zusätzlich zu diesem Datensatz ein Klassifikator X, welcher der Multiklassifikation fähig ist, an CT übergeben. Die weiteren Spezifikationen des Klassifikators X und die konkreten Attribute des Datensatzes sind für CT nicht relevant, da der Klassifikator als Blackbox behandelt und die Attribute einzelner Instanzen nicht betrachtet werden.

Zuerst wird der Startbaum erzeugt. Dieser besteht aus einer Wurzel und einem Blatt für jede Klasse im Datensatz. Abbildung 3 zeigt den Startbaum.

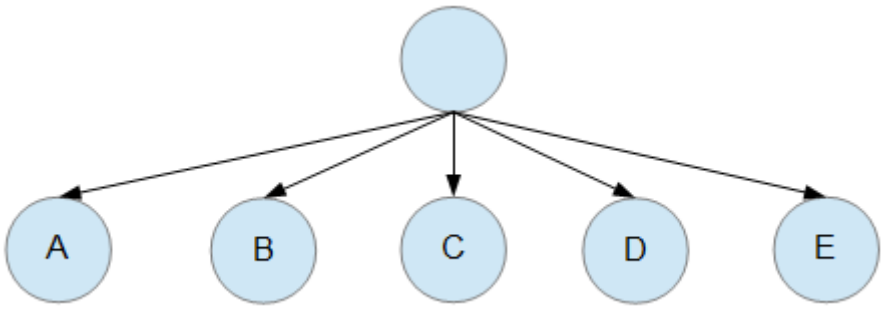


Abbildung 3: Baum vor der ersten Iteration

Nun beginnt die erste Iteration. Für den Datensatz wird der Klassifikator gelernt, anschließend mit dem Datensatz getestet und die Konfusionsmatrix erzeugt, welche in der folgenden Tabelle zu sehen ist.

/	A	B	C	D	E
A	20	2	0	3	1
B	1	24	0	4	1
C	1	0	21	2	5
D	3	5	1	21	2
E	0	2	6	3	17

Die größte Überlappung liegt zwischen den Klassen C und E mit elf vor, weil fünf Instanzen der Klasse C als E klassifiziert wurden und sechs Instanzen der Klasse E als Klasse C. Es wird ein neuer Datensatz erzeugt, der nur die Instanzen enthält, welche zu den Klassen C und E gehören. Der ursprüngliche Datensatz wird dahingehend modifiziert, dass alle Instanzen der Klassen C und E nun der Klasse CE angehören.

Klasse	A	B	CE	D
Anzahl	26	30	57	32

Es wird der Knoten CE erzeugt und ein Klassifikator gelernt aus dem Datensatz, der nur die Instanzen der Klassen C und E enthält. Der Knoten wird anschließend in den Baum eingefügt. Abbildung 4 zeigt den Baum nach der ersten Iteration.

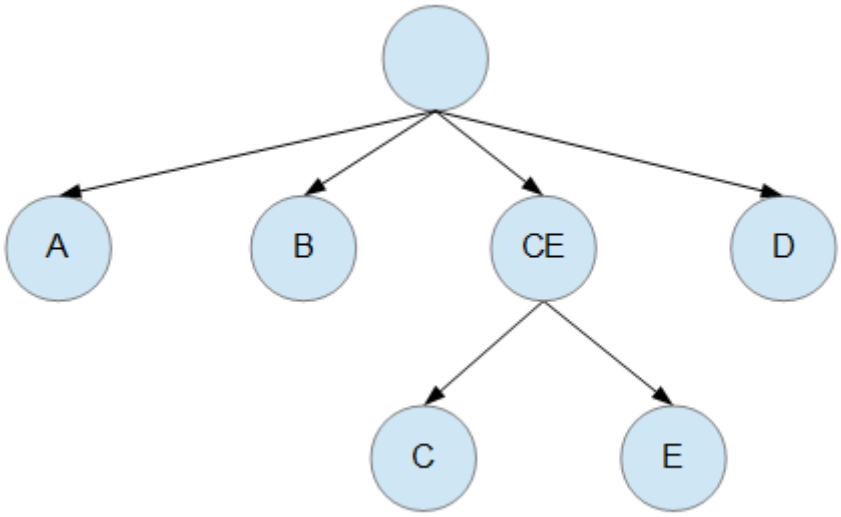


Abbildung 4: Baum nach der ersten Iteration

Anschließend startet die zweite Iteration mit dem Lernen des Klassifikators aus dem modifizierten Datensatz mit nur noch vier Klassen und der Erzeugung der Konfusionsmatrix.

/	A	B	CE	D
A	20	2	1	3
B	1	24	1	4
CE	1	2	49	5
D	3	5	3	21

Hier besteht die größte Überlappung zwischen den Klassen B und D mit neun. Wie zuvor wird ein neuer Datensatz durch Zusammenfassen dieser beiden Klassen erzeugt. Für den neuen Metaknoten wird weiterhin ein Klassifikator aus einem Datensatz, der nur Instanzen der Klassen B und D enthält, gelernt.

Klasse	A	BD	CE
Anzahl	26	62	57

In der Konfusionsmatrix sind die Werte der Klassen, welche in diesem Beispiel fusioniert werden, der Einfachheit halber zusammengefasst. In der Praxis ist dies sehr stark vom Datensatz und Klassifikator abhängig und keineswegs immer der Fall.

Abbildung 5 zeigt den Baum nach der zweiten Iteration.

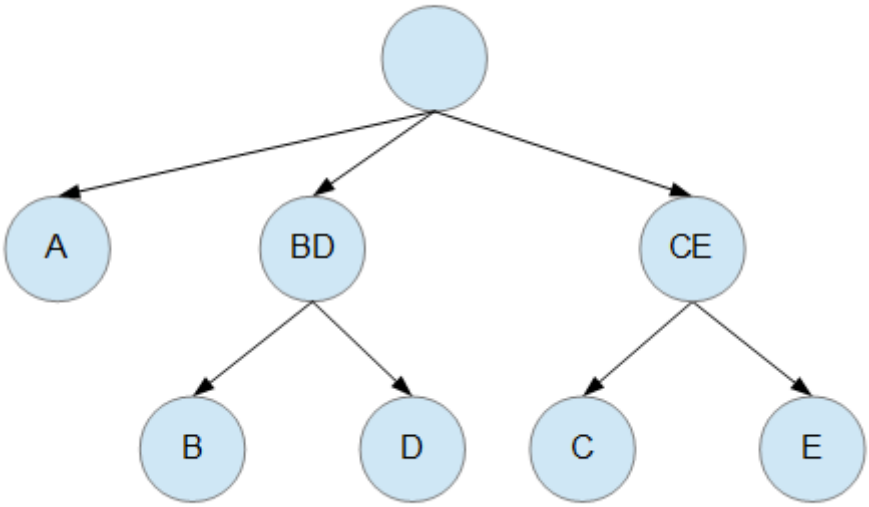


Abbildung 5: Baum nach der zweiten Iteration

Damit beginnt die dritte und letzte Iteration. Die Konfusionsmatrix sieht wie folgt aus.

/	A	BD	CE
A	20	5	1
BD	4	54	4
CE	1	7	49

Hier besteht die größte Überlappung zwischen den Klassen BD und CE mit elf. Wie in den Iterationen davor werden diese beiden Klassen zusammengefasst, ein neuer Metaknoten BDCE erzeugt und für diesen der Klassifikator gelernt. Damit ist der Baum binär und der Algorithmus endet. Da der Algorithmus nicht vorzeitig abgebrochen ist, muss für die Wurzel nochmals ein Klassifikator gelernt werden. In Abbildung 6 ist der finale Baum zu sehen.

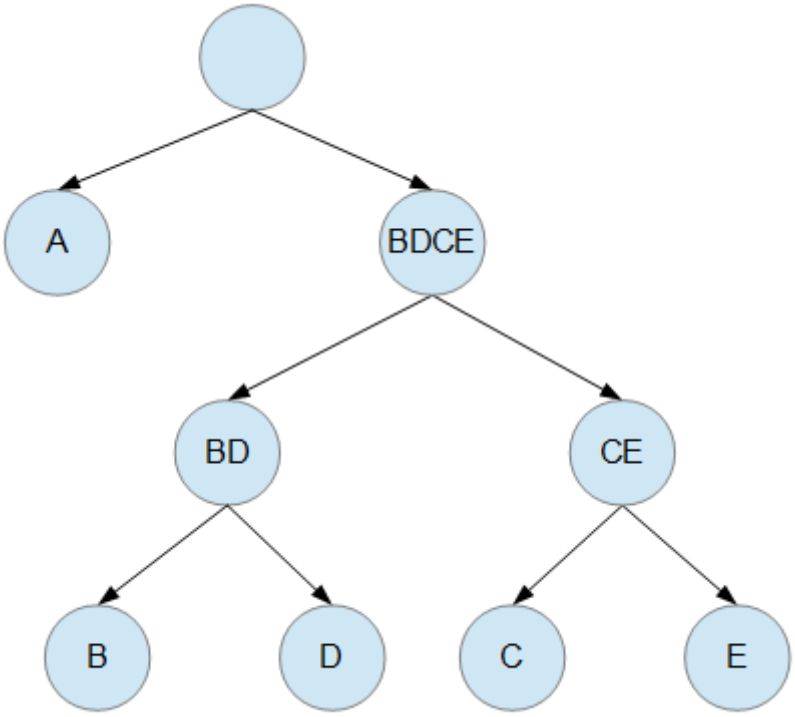


Abbildung 6: Baum nach der dritten Iteration

3.3 Implementation

In diesem Abschnitt wird beschrieben, wie CT mithilfe des weka Frameworks in Java implementiert wurde. Die Vorgaben von weka zur Implementierung eines eigenen Klassifikators wurden dabei eingehalten. Nachfolgend werden die zwei relevanten Funktionen buildClassifier() und classifyInstance() beschrieben, welche jeweils den Klassifikator erzeugen bzw. eine Instanz klassifizieren.

Für die Beschreibung des Baumes wurde die Klasse ICTNode erstellt. Ein Element dieser Klasse repräsentiert einen Knoten im Baum. In jedem Knoten kann jeweils ein Klassifikator gespeichert werden. Weiterhin besitzt jeder Knoten Verknüpfungen zu seinen Kindern und in jedem Knoten ist gespeichert, welche Klasse er repräsentiert, die Tiefe des durch ihn induzierten Subbaumes und ob es sich bei ihm um ein Blatt handelt.

Zu Beginn wird der Startbaum mit Wurzelknoten und sämtlichen Blattknoten gebildet. In jeder Iteration werden ein Klassifikator auf dem Datensatz gelernt und die Konfusionsmatrix erzeugt. Um die zwei Klassen auszuwählen, welche zusammengefasst werden sollen, wird in einer doppelten Schleife über die obere Dreiecksmatrix der Konfusionsmatrix iteriert. Jeder Schritt überprüft, ob die zwei aktuell ausgewählten Klassen nach vorgegebenen Parametern optimaler wären als die zuvor gefundene Kombination. Trifft dies zu, werden die beiden aktuellen Klassen als Optimum gesetzt.

Sind die beiden Klassen mit der größten Überlappung gefunden, werden zwei neue Datensätze erzeugt. Der erste Datensatz enthält alle Instanzen des alten, allerdings werden die Klassen jener

Instanzen angepasst, welche den zwei ermittelten Klassen angehören. Der zweite Datensatz beinhaltet ausschließlich die Instanzen dieser beiden Klassen. In weka wird dies realisiert, indem zunächst ein leerer Datensatz mit den Spezifikationen des alten erzeugt wird. Die anschließende Iteration über die Instanzen ändert die Klassen jener, bei denen es notwendig ist, und fügt die Instanzen dem Datensatz hinzu. Hier ist anzumerken, dass bei diesem Vorgang keine neue Klasse erzeugt wird, sondern alle Instanzen beider Klassen auf die Klasse mit dem kleineren Wert von beiden gesetzt werden. Dies hat keine Auswirkungen auf das Endergebnis, ist einfacher umzusetzen und spart Ressourcen. Anschließend wird ein neuer Knoten erzeugt, welcher diese beiden zusammengefügten Klassen repräsentiert, an die Wurzel gehangen und die Knoten der beiden Klassen wiederum an ihn. Für den Datensatz des Metaknotens wird der Klassifikator gelernt und im Metaknoten gespeichert.

Wenn kein verfrühter Abbruch stattfand, muss nach der letzten Iteration noch einmal ein Klassifikator für den Wurzelknoten gelernt werden. Da der Klassifikator zu Beginn der Iteration erzeugt wurde, um die Konfusionsmatix zu berechnen, ist dieser am Ende der Iteration aufgrund der geänderten Kindknoten nicht mehr aktuell.

Es ist auf zwei weitere Parameter hinzuweisen, welche jedoch das Ergebnis selbst nicht beeinflussen. Der erste Parameter -dC steht für “delete Classes“. Ist dieser gegeben, werden bei der zuvor beschriebenen Erzeugung der zwei Datensätze die gelöschten Klassen jeweils aus den Spezifikationen der Datensätze entfernt. Im Standardfall wird dies aufgrund des entstehenden Mehraufwandes nicht durchgeführt. Für Naive Bayes ist dieser Schritt allerdings zwingend notwendig, da das Ergebnis wegen der dort verwendeten Laplace-Verschiebung sonst verfälscht werden oder es zu Abstürzen im weka Framework kommen würde. Wenn ein Klassifikator eine Laplace-Verschiebung oder eine ähnliche Methode benutzt, muss dieser mit dem zusätzlichen Parameter -dC aufgerufen werden. Dieser Parameter ist jedoch nicht Standard, weil das Löschen von Klassen aus der Datensatzspezifikation zum Beispiel bei J48 zu Abstürzen von weka führt, zusätzlich zu dem oben genannten unnötigen Mehraufwand.

Der zweite Parameter ist -dT, was für “draw Tree“ steht. Ist dieser gegeben, wird zusätzlich eine Visualisierung des erzeugten Baumes ausgegeben.

Um eine Instanz zu klassifizieren, wird auf dem Wurzelknoten des Baumes die rekursive Funktion classifyInstance() aufgerufen. Handelt es sich bei dem Knoten um ein Blatt, wird die repräsentierte Klasse zurückgegeben. Andernfalls wird die gegebene Instanz mit dem Klassifikator des Knotens klassifiziert und classifyInstance() auf dem Kindknoten aufgerufen, welcher die vorhergesagte Klasse repräsentiert.

4 Experimentelle Evaluierung

4.1 Testspezifikationen

In diesem Kapitel werden die Ergebnisse der Anwendung von CT auf verschiedene Datensätze und Klassifikatoren vorgestellt und diskutiert. In den folgenden vier Abschnitten werden die Ergebnisse der einzelnen Parameter präsentiert und es wird diskutiert, wie sich diese zum ursprünglichen Klassifikator und zu den anderen Parametern verhalten. Für sämtliche Testläufe wurde 10-fache “Cross Validation“ eingesetzt. Die Testläufe wurden alle in der in Abschnitt 3.3 beschriebenen Implementierung durchgeführt. Nachfolgend sind die verwendeten Datensätze aufgelistet:

Tabelle 1: Die einzelnen verwendeten Datensätze mit ihren Merkmalen

Name	# Instanzen	# Klassen	# Attribute	default Error
segment-challenge	1500	7	20	15.73%
soybeans	683	19	36	13.47%
letter	20000	26	17	4.07%
abalone	4177	28	9	16.50%
car	1728	4	7	70.02%
glass	214	7	10	35.51%
Page-blocks	5473	5	11	89.77%
Solar-flare-c	1712	8	11	85.16%
Solar-flare-m	1389	9	11	95.10%
thyroid_hyper	3772	5	30	97.30%
thyroid_hypo	3772	5	30	92.29%
thyroid_rep	3772	5	30	96.71%
vehicel	846	4	19	25.77%
vowel	990	11	14	9.09%
yeast	1484	10	9	31.20%
landsat	4435	6	37	24.17%
optdigit	5620	10	65	10.18%

Als Klassifikatoren wurden Naive Bayes, J48 / C4.5, SMO, Random Forest und JRip / Ripper verwendet. Sie alle unterscheiden sich strukturell in ihrer Funktionsweise voneinander und werden jeweils in der Praxis für verschiedene Anwendungen verwendet. Somit stellen sie eine differenzierte Auswahl für den Test von CT hinsichtlich seiner Wirkung auf verschiedene Klassifikatoren.

Die jeweiligen Klassifikatoren wurden in den Testläufen in ihrer weka Standardkonfiguration eingesetzt. Tabelle 2 zeigt die Ergebnisse, welche als Vergleichsgrundlage für alle Testläufe dienten und das Maß waren, ob CT eine Verbesserung darstellte oder nicht.

Mit diesen siebzehn Datensätzen erreichte Naive Bayes eine Durchschnittsgenauigkeit von 75,20%, J48 von 83,36%, SMO von 81,90%, Random Forest von 85,92% und JRip von 81,96%.

Tabelle 2: Genauigkeit der Standardklassifikatoren auf den verwendeten Testdatensätzen in Prozent

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	81.07	95.73	91.93	96.67	93.13
soybeans	92.97	91.51	93.85	91.22	92.24
letter	64.11	87.98	82.34	93.97	86.04
abalone	23.84	21.16	25.26	23.08	19.37
car	85.53	92.36	93.75	93.98	86.46
glass	48.60	66.82	56.07	74.30	68.69
Page-blocks	90.85	96.88	92.93	97.20	96.84
Solar-flare-c	79.91	85.11	85.16	84.40	85.40
Solar-flare-m	90.78	95.10	95.10	93.45	94.74
thyroid_hyper	95.39	98.91	97.83	98.67	98.49
thyroid_hypo	95.25	99.55	93.58	99.07	99.34
thyroid_rep	93.11	99.13	96.74	98.57	98.94
vehicel	44.80	72.46	74.35	76.00	69.03
vowel	63.74	81.52	71.41	94.65	70.10
yeast	57.61	56.00	57.08	58.76	58.09
landsat	79.50	86.16	86.56	89.92	85,64
optdigit	91.33	90.69	98.33	96.74	90.71

Die Verbesserungen bzw. Verschlechterungen, die durch CT erzielt wurden, müssen unter Berücksichtigung dieser Werte betrachtet werden. Offensichtlich ist eine 2%-ige Verbesserung auf einem Klassifikator mit 80%-iger Genauigkeit von größerem Wert als auf einem mit 60%.

Die in den folgenden Abschnitten genannten durchschnittlichen Verbesserungen oder Verschlechterungen der Genauigkeit sind immer in ihrem Kontext zu betrachten. Ein höherer Durchschnitt bedeutet nicht unbedingt, dass auch das Gesamtergebnis besser ist. Häufig existieren einzelne Datensätze, auf denen der Parameter ein besonders gutes Ergebnis lieferte und damit den Durchschnitt deutlich erhöhte. Ebenso war häufig zu beobachten, dass der Durchschnitt bei einem Parameter im Vergleich zu anderen geringer war, allerdings die Anzahl der Datensätze, welche eine Verbesserung aufwiesen, stieg. Die Frequenz der durch CT bewirkten Verbesserungen ist ein ebenso wichtiger Punkt, wie deren Größe.

Für die Erzeugung der in Abschnitt 4.6 dargestellten Bäume wurde keine 10-fache “Cross Validation“ eingesetzt, sondern die Trainingsmenge als Testmenge verwendet. Dadurch spiegeln die abgebildeten Bäume die Ergebnisse nicht exakt wieder. Bei 10-facher “Cross Validation“ würden entsprechend zehn Bäume entstehen, die sich allerdings nicht oder kaum voneinander unterscheiden würden. Die strukturellen Merkmale der Bäume blieben unverändert.

4.2 Absolute Überlappung

Dieser Abschnitt diskutiert die Ergebnisse bei absoluter Überlappung. Dies ist die Standardkonfiguration von CT und - wie in Kapitel 3.1 beschrieben - werden in jeder Iteration die beiden Knoten zusammengefasst, bei denen die Summe von “false negatives“ und “false positives“ den absolut höchsten Wert hat. Es werden nacheinander die Ergebnisse der einzelnen Klassifikatoren diskutiert.

Tabelle 3 zeigt die Unterschiede zu den fünf ursprünglichen Klassifikatoren für die siebzehn Testdatensätze. Die Effektivität von CT in der Standardversion ist gemischt. Eine Tabelle 9 mit den absoluten Werten ist im Anhang A.1.1 zu finden. Weiterhin befinden sich dort Grafiken für die hier nicht gezeigten Klassifikatoren.

Tabelle 3: Differenz der Standardvariante von CT zu den ursprünglichen Klassifikatoren auf den Testdatensätzen

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-5,07	0,40	0,87	0,67	2,20
soybeans	-9,66	0,59	-0,29	-0,29	1,02
letter	-13,22	-0,12	-13,09	-2,77	2,19
abalone	-11,44	4,24	-3,97	0,67	6,92
car	-1,22	-0,69	-1,56	0	0,87
glass	-9,81	3,74	0	1,40	0,93
Page-blocks	-1,97	0,15	0,64	-0,11	-0,07
Solar-flare-c	0,29	0,47	0	-0,23	-0,29
Solar-flare-m	-0,58	0	0	0,14	0
thyroid_hyper	0,03	-0,08	0,05	0,03	0,05
thyroid_hypo	-0,05	-0,07	0,27	-0,21	-0,08
thyroid_rep	-0,16	-0,03	-0,03	-0,05	0,21
vehicel	-4,26	-2,36	-0,24	-0,71	0,35
vowel	-5,25	-0,81	2,83	-0,40	6,77
yeast	-0,54	2,83	-5,39	-1,21	-0,88
landsat	-2,53	-1,01	0,50	-0,56	1,10
optdigit	-5,14	0,28	-1,44	-2,28	1,62

Für Naive Bayes sind die Ergebnisse unbefriedigend. Auf den 17 Datensätzen ist CT mit Naive Bayes im Schnitt 4,15% ungenauer als der ursprüngliche Klassifikator. Das schlechteste Ergebnis erzielte letter mit einer Verschlechterung um 13,22%. Das beste Ergebnis hatte solar-flare-c mit einer Verbesserung von 0,29%. Insgesamt war nur auf zwei der 17 Datensätze eine Verbesserung zu verzeichnen. Eine Anwendung der Standardversion von Classifer Trees mit Naive Bayes ist aus diesen Gründen nicht zu empfehlen.

Die Ergebnisse für J48 sind besser als für Naive Bayes. Im Durchschnitt ist eine Verbesserung von 0,44% gegenüber dem ursprünglichen Klassifikator zu beobachten. Dabei wird die größte Verbesserung bei abalone erreicht mit 4,24%, die größte Verschlechterung bei vehicel mit -2,36%. Bei acht Datensätzen gab es eine Verbesserung, bei weiteren acht eine Verschlechterung

und bei einem keinen Unterschied. Wie in Abbildung 7 zu sehen, erzielt CT bei den drei Datensätzen abalone, glass und yeast signifikante Verbesserungen. CT auf J48 anzuwenden erscheint sinnvoll, wenn man bereit ist, eine längere Erzeugungsdauer des Klassifikators zu akzeptieren.

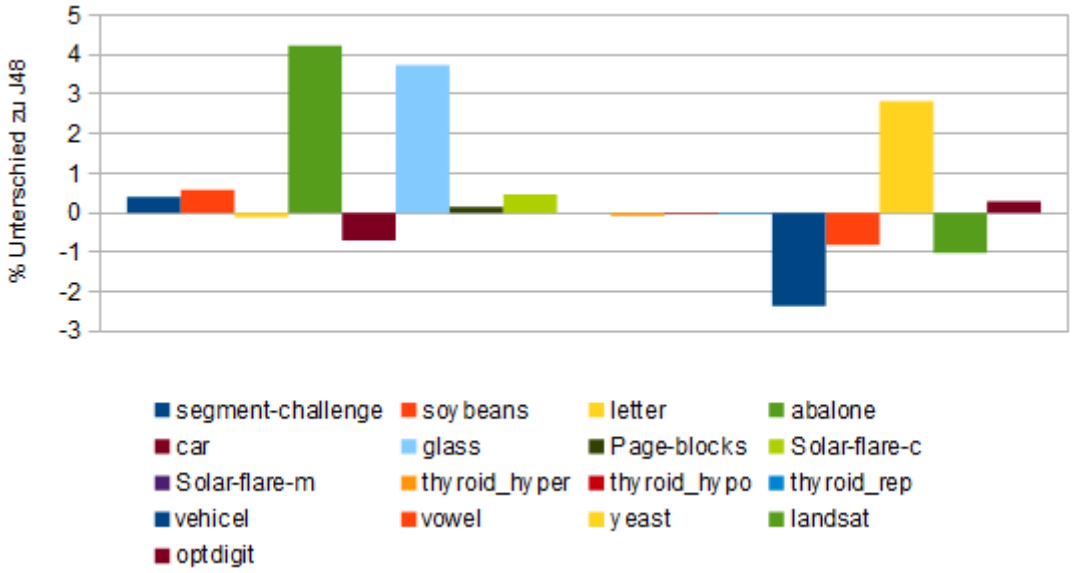


Abbildung 7: Genauigkeitsunterschiede von J48 zu CT mit J48 bei -A

Für SMO bewirkt CT im Durchschnitt einen Genauigkeitsverlust von 1.23%. Darunter stechen einige wenige Datensätze im Vergleich zum Original negativ hervor, wie letter mit -13,09%. Insgesamt wird für sechs Datensätze eine Verbesserung erreicht, wobei die höchste für vowel bei 2,83% liegt und somit ein gutes Ergebnis erzielt. Die Anwendung von CT auf SMO kann wie bei J48 sinnvoll sein, jedoch in weniger Fällen.

Random Forest erzielt einen leicht negativen Durchschnitt von -0,35%. Die Differenzen der Genauigkeiten sind allgemein geringer, weil Random Forest insgesamt genauer ist als die anderen vier Klassifikatoren. Bei fünf Datensätzen war eine Verbesserung beobachtbar, die größte bei glass mit 1,40%. Zehn Datensätze verzeichneten eine Verschlechterung, mit dem größten Unterschied bei letter mit -2,77%. Insgesamt empfiehlt sich CT bei Random Forest ebenso häufig wie bei SMO.

Bei JRip erreicht CT das beste Ergebnis mit einer durchschnittlichen Verbesserung von 1,35%. Abbildung 8 zeigt eine Visualisierung der Ergebnisse. Bei nur vier Datensätzen ist das Ergebnis schlechter, mit dem größten Unterschied bei yeast mit -0,88%. Bei zwölf Datensätzen gab es eine Verbesserung, mit der größten bei abalone und vowel mit 6,92% und 6,77%. Für neun der zwölf Datensätze ist eine Steigerung von über einem halben Prozent zu verzeichnen, was für mehr als die Hälfte der Datensätze eine signifikante Verbesserung bedeutet.

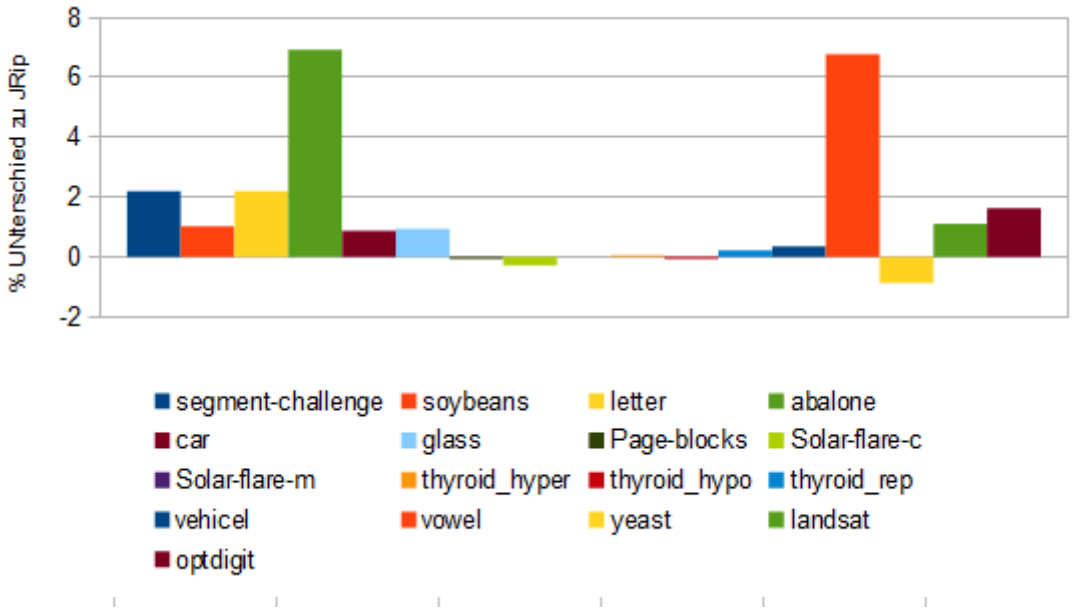


Abbildung 8: Genauigkeitsunterschiede von JRip zu CT mit JRip bei -A

4.3 Relative Überlappung (-R)

In diesem Abschnitt werden die Ergebnisse der relativen Überlappung diskutiert und mit denen aus dem vorherigen verglichen. Wie in Kapitel 3.1 beschrieben, werden die zwei zusammenzufassenden Klassen bestimmt, indem die Summe von “true positives“ und “true negatives“ durch die Summe von “false positives“ und “false negatives“ geteilt und der niedrigste Wert ausgewählt wird. Tabelle 4 zeigt die Unterschiede zu den ursprünglichen Klassifikatoren. Die absoluten Werte zu den einzelnen Klassifikatoren sind in Tabelle 10 in Anhang A.1.2 aufgeführt. Dort sind auch die hier nicht gezeigten Grafiken der einzelnen Klassifikatoren zu finden.

Für Naive Bayes ist das Ergebnis besser als bei absoluter Überlappung, allerdings mit einer durchschnittlichen Verschlechterung von -2,97% immer noch weit von der Genauigkeit des ursprünglichen Klassifikators entfernt. Ebenfalls ist nur für zwei der Datensätze eine Verbesserung sichtbar, jedoch fallen diese deutlicher aus: bei car mit 1,79% und page-blocks mit 1,55%. Mit -A erreichten diese beiden Datensätze circa -3%. Bei dem Datensatz abalone steigt das Ergebnis mit -R um über 11%. Der größte Verlust wurde bei letter mit -12,72% beobachtet. Es wird deutlich, dass verschiedene Parameter auf denselben Datensätzen sehr unterschiedliche Ergebnisse bewirken können. Insgesamt erzielen zehn Datensätze mit -R ein besseres Ergebnis, fünf mit -A. Während für Naive Bayes -A nicht sinnvoll erscheint, erzielt die Anwendung von -R zumindest in einigen wenigen Fällen wünschenswerte Resultate.

Für J48 wird mit -R eine durchschnittliche Steigerung von 0,38% erzielt. Für zehn Datensätze, zwei mehr als bei -A, wird eine Verbesserung erreicht, mit dem höchsten Wert bei abalone mit 4,09%. vehicel erzielt mit -2,36% das schlechteste Ergebnis. Ähnlich wie bei Naive Bayes unterscheiden sich die Ergebnisse von -R und -A bei einigen Datensätzen erheblich, zum Beispiel glass ist bei -A um 3,27% besser als -R. Es ist festzustellen, dass -A eine höhere Amplitude und -R eine höhere Frequenz bezüglich der Verbesserungen erzielt. Im direkten Vergleich war -A bei

Tabelle 4: Differenz der relativen Variante (-R) von CT zu den ursprünglichen Klassifikatoren auf den Testdatensätzen

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-5,07	0,27	0,87	0,67	2,20
soybeans	-8,49	0,73	-0,44	-0,15	0,88
letter	-12,72	-0,58	-9,73	-2,45	1,66
abalone	-0,34	4,09	0,19	0,38	6,73
car	1,79	1,50	-0,64	0,69	2,66
glass	-9,35	0,47	1,87	2,80	-1,40
Page-blocks	1,55	0,11	0,09	0,02	-0,29
Solar-flare-c	-2,69	0,41	0	-0,06	-0,29
Solar-flare-m	-0,72	0	0	-0,07	-0,14
thyroid_hyper	-0,37	-0,05	0,08	0,05	0
thyroid_hypo	-0,58	-0,05	0,27	-0,13	-0,08
thyroid_rep	0	0,05	-0,027	-0,05	0,19
vehicel	-2,25	-2,36	-0,24	-0,71	0,35
vowel	-2,93	-1,11	6,77	-0,10	6,06
yeast	-1,08	2,76	-1,48	-0,94	-0,47
landsat	-2,19	-0,45	0,65	-0,41	1,01
optdigit	-5,14	0,66	-1,58	-0,82	1,90

neun Datensätzen besser, -R bei sechs. Insgesamt ist die Differenz der Werte zwischen -A und -R geringer als bei Naive Bayes.

SMO erzielt mit -R eine durchschnittliche Verschlechterung von 0,20% gegenüber dem ursprünglichen Klassifikator und damit ein um knapp ein Prozent besseres Ergebnis als mit -A. Insgesamt war in acht Fällen, damit zwei mehr als bei -A, eine Verbesserung zu beobachten, mit dem höchsten Wert bei vowel mit 6,77%. Bei den sieben Verschlechterungen erzielte letter mit -9,73% den niedrigsten Wert. Wie bei -A stechen diese beiden Datensätze hervor, bei den anderen ist der Unterschied wesentlich geringer. Bei SMO erzielten in sechs Fällen -A und -R dasselbe Ergebnis, insbesondere bei einfach zu klassifizierenden Datensätzen. Bei nur drei Datensätzen konnte -A ein besseres Ergebnis als -R erzielen und davon nur in einem Fall mit einem Wert größer als ein halbes Prozent, damit kann -A aufgrund der Dominanz von -R für SMO vernachlässigt werden.

Für Random Forest wird eine Steigerung von -0,35% bei -A auf -0,07% mit -R erreicht. Insgesamt findet eine Verbesserung bei sechs Datensätzen, einem mehr als bei -A, mit dem höchsten Wert bei glass mit 2,80% statt. Bei den anderen elf war eine Verschlechterung zu verzeichnen, mit dem niedrigsten Wert bei letter mit -2,45%. Insgesamt unterscheiden sich die Ergebnisse von -A und -R wenig, allerdings häufig zu Gunsten von -R. Auf nur zwei Datensätzen war -A besser, für drei waren beide Varianten gleich stark. Ähnlich zu SMO wird auch bei Random Forest -A von -R dominiert.

Ähnlich zu J48 schneidet -R mit JRip mit einer durchschnittlichen Verbesserung von 1,23% schlechter ab als -A. Im Vergleich zu -A steigt die Anzahl der Datensätze, die schlechter als der

ursprüngliche Klassifikator abschneiden, von vier auf sechs, mit dem niedrigsten Wert bei glass mit -1,40%. Die besten Werte erzielen wie bei -A abalone mit 6,73% und vowel mit 6,06%. Insgesamt weisen acht der Datensätze eine Verbesserung von über einem halben Prozent auf. Auf nur drei Datensätzen war -R besser als -A und auf vier gleich stark. Die Differenzen lagen bei drei Datensätzen unter einem halben Prozent. Abbildung 9 veranschaulicht diese Ergebnisse. Für JRip ist damit im Gegensatz zu Naive Bayes, SMO und Random Forest -A stärker.

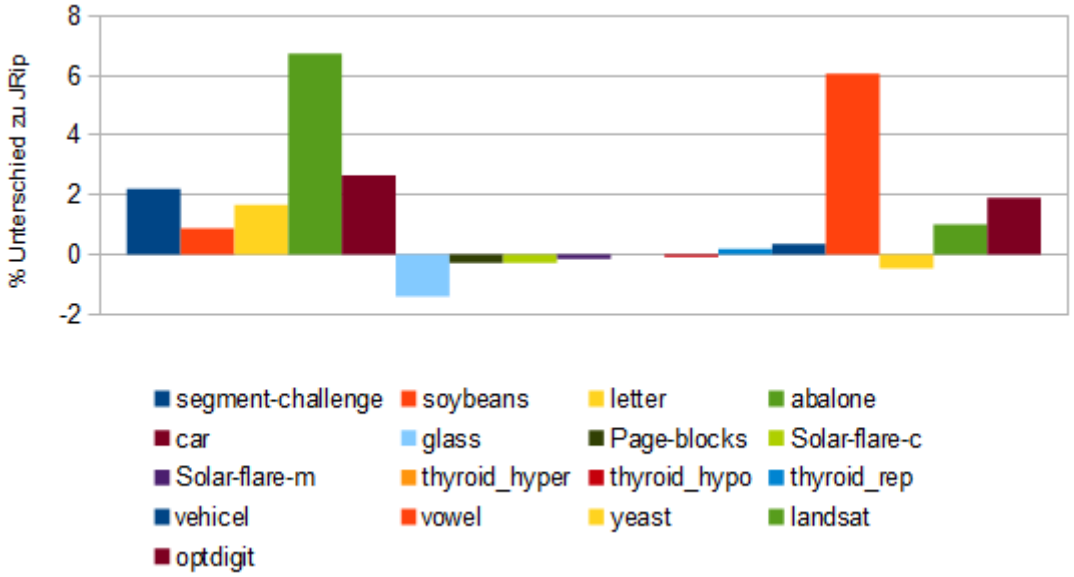


Abbildung 9: Genauigkeitsunterschiede von JRip zu CT -R mit JRip

Ein weiterer Vergleich von -A und -R fand in Bezug auf die Datensätze statt. Die Fragestellung war, ob es Datensätze gibt, auf denen -A oder -R unabhängig vom Klassifikator stärker ist. Für jeden Klassifikator wurden die einzelnen Datensätze in drei Kategorien eingeordnet. Die Kategorien sind “-A besser“, “-R besser“ und “<0,1%“, mit der Bedeutung, dass die Differenz beider Ergebnisse unter diesem Grenzwert liegt.

Der einzige Datensatz, der nur in einer Kategorie auftaucht, ist car in “-R besser“. In “-R besser“ und “<0,1%“ gleichzeitig enthalten sind thyroid_rep, vehicel und landsat, in “-A besser“ und “<0,1%“ segment-challenge, Solar-flare-m, thyroid_hyper und thyroid_hypo. thyroid_rep, thyroid_hyper, thyroid_hypo und segment-challenge fanden sich für vier Klassifikatoren in der Kategorie “<0,1%“. Diese vier Datensätze sind einfach zu klassifizieren, etwa durch einen hohen default Error oder wenig Ausreißer in den Daten. Dadurch lässt sich erklären, warum diese Datensätze in den genannten Kategorien stehen. Solar-flare-m fällt in eine ähnliche Gruppe wie diese vier, auch wenn der Datensatz in nur zwei Fällen in der Kategorie <0,1% erschien, da die größte Abweichung maximal 0,22% betrug.

Interessant sind die beiden Datensätze car und vehicel. Beide bestehen aus nur 4 Klassen, unterscheiden sich aber stark im default Error. Die Instanzen in vehicel sind nahezu über alle Klassen gleich verteilt, bei car befinden sich über 70% in einer Klasse. Für vier Klassifikatoren ist das Ergebnis für vehicel bei -A und -R gleich. Infolge der wenigen Klassen und der Gleichverteilung der Instanzen verhalten sich -A und -R nahezu identisch. Dadurch entstehen nie lange Ketten und in der Folge treten Unterschiede in der Anzahl von Instanzen in einzelnen inneren Knoten

nicht auf. Bei car reicht der Vorteil von -R gegenüber -A auf den einzelnen Klassifikatoren von 0,69% bis 3,01% und ist damit deutlich. Der ähnlichste Datensatz zu car in Bezug auf die Anzahl der Klassen und default Error ist Page-blocks mit 5 Klassen und knapp 90% default Error. Jedoch war bei diesem Datensatz das bei car festgestellte Verhalten nicht zu beobachten. In Bezug auf diese beiden Merkmale war landsat der nächst ähnliche Datensatz zu car, bei dem eine Tendenz zu -R auftrat, allerdings fiel diese nicht so stark aus wie bei car. Bei Datensätzen mit einer höheren Anzahl von Klassen war keine Tendenz zu beobachten.

Es wird geschlussfolgert, dass bei Datensätzen mit wenig Klassen und der daraus folgenden geringen Anzahl von Iterationen Gemeinsamkeiten zwischen -A und -R auftreten können. Mit steigender Anzahl der Klassen relativieren sich die Gemeinsamkeiten, da sich die Eigenschaften der Klassifikatoren stärker auswirken.

4.4 Absolute Balance (-aB)

In diesem Kapitel werden die Ergebnisse des Parameters -aB vorgestellt und diskutiert. Tabelle 5 zeigt die Unterschiede zu den ursprünglichen Klassifikatoren mit dem Parameter -aB und Tabelle 6 die Unterschiede mit den Parametern -R -aB. Die Tabellen 11 und 12 enthalten die absoluten Werte und befinden sich im Anhang A.1.3, hier nicht gezeigte Grafiken der Ergebnisse der einzelnen Klassifikatoren ebenfalls.

Tabelle 5: Differenz der absoluten Variante mit absoluter Balance (-aB) von CT zu den ursprünglichen Klassifikatoren auf den Testdatensätzen

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-6,87	0,33	0,80	0,20	2,47
soybeans	-0,59	1,02	-0,59	-0,44	0,29
letter	-6,16	-1,28	-7,07	-0,24	0,68
abalone	-0,86	2,32	0,29	0,79	6,27
car	-0,46	0,12	-0,52	-0,23	1,50
glass	-1,87	-0,47	2,80	2,34	0
Page-blocks	-2,05	0,16	0,58	0,07	0,13
Solar-flare-c	-0,23	0,47	0	-0,18	-0,29
Solar-flare-m	-0,36	0	0	0,14	0
thyroid_hyper	-0,21	-0,08	0,05	-0,05	0
thyroid_hypo	-0,05	-0,03	0,27	-0,11	0,05
thyroid_rep	-0,11	-0,07	0	0,05	0,11
vehicel	-2,25	-2,25	-4,37	-1,18	0,83
vowel	-3,43	-6,26	-3,03	-0,61	6,46
yeast	-0,81	1,08	-1,95	-0,27	0
landsat	-1,80	-0,43	0,14	-0,02	0,72
optdigit	-10,09	-0,71	-1,21	-0,44	0,94

Im Vergleich zu -A und -R bewirkte die Anwendung von -aB bei Naive Bayes eine leichte Steigerung in der Durchschnittsgenauigkeit gegenüber dem ursprünglichen Klassifikator und erzielte

Tabelle 6: Differenz der relativen Variante mit absoluter Balance (-R -aB) von CT zu den ursprünglichen Klassifikatoren auf den Testdatensätzen

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-6,87	0,53	0,80	0,20	2,27
soybeans	-0,59	1,02	-0,59	-0,15	0,44
letter	-11,05	-1,41	-7,46	-0,39	0,62
abalone	-0,38	2,39	-1,13	0,57	5,75
car	1,22	0,12	-0,64	0,81	2,08
glass	-3,27	0	3,27	3,74	-2,80
Page-blocks	2,25	0,22	0,58	0,13	0,11
Solar-flare-c	-0,76	0,41	0	-0,18	-0,29
Solar-flare-m	-0,36	0	0	0	0,07
thyroid_hyper	-0,27	-0,05	0,05	0	-0,027
thyroid_hypo	-0,61	-0,05	0,27	-0,13	0,05
thyroid_rep	-0,05	0,05	0	0	0,11
vehicel	-2,25	-2,25	-4,37	-1,18	0,83
vowel	-4,34	-0,71	-0,10	0,10	6,77
yeast	-1,28	1,01	-0,61	-0,54	0,34
landsat	-1,80	-0,61	-0,32	-0,05	0,32
optdigit	-10,09	-0,36	-1,16	0	0,96

mit -aB nur einen Genauigkeitsverlust von 2,25%, mit -R -aB von 2,38%. Für -aB war bei jedem der 17 Datensätze eine Verschlechterung der Genauigkeit im Vergleich zum ursprünglichen Klassifikator zu beobachten. Demzufolge ist eine Anwendung von -aB für Naive Bayes nicht zu empfehlen. Bei -R -aB zeigten zwei Datensätze eine Verbesserung: car mit 1,22% und Page-blocks mit 2,25%. Damit stellt -R -aB eine Alternative gegenüber -R dar, die Ergebnisse von CT mit Naive Bayes bleiben auch mit Anwendung von -aB hinter denen der anderen getesteten Klassifikatoren zurück.

Mit Nutzung von -aB sank für J48 die Durchschnittsgenauigkeit: -aB erzielte -0,35% und -R -aB 0,02%. Eine Verbesserung war bei sieben Datensätzen für -aB und bei acht für -R -aB zu verzeichnen und damit insgesamt bei weniger Datensätzen als bei -A und -R. Zusätzlich waren die Steigerungen der einzelnen Werte niedriger als im Vergleich zu den Tests ohne -aB. Mit -aB erzielte abalone mit 2,32% den höchsten Wert, mit -R -aB ebenfalls abalone mit 2,39%. Andere Datensätze erreichten circa 1% als maximalen Wert, während bei -A und -R mehrere Datensätze diese Grenze deutlich überschritten. Damit ist -aB aufgrund der genannten Ergebnisse für J48 nicht zu empfehlen.

SMO erzielte mit -aB eine Durchschnittsgenauigkeit von -0,81% im Vergleich zu dem ursprünglichen Klassifikator, -R -aB -0,67%. Für -aB wurde eine Verbesserung auf sieben Datensätzen erreicht, bei -R -aB nur auf fünf. Die besten Werte wurden jeweils bei glass erzielt mit 2,80% für -aB und 3,27% für -R -aB. Somit lagen die Ergebnisse dieser beiden Parameter zwischen denen von -A und -R, wobei -R -aB ein besseres Ergebnis lieferte als -aB. Analog zu dem Vergleich zwischen -A und -R waren die Ergebnisse unter zusätzlicher Nutzung von -aB sogar für neun

der Datensätze identisch. Auffallend ist, dass es sich bei diesen um die einfacher zu klassifizierenden handelt. Trotz der durchschnittlich schlechteren Ergebnisse im Vergleich zu -R kann sich die Nutzung von -aB oder -R -aB bei SMO auszahlen, weil nicht identische Teilmengen der Datensätze gute Ergebnisse erzielen.

Random Forest erzielte unter Nutzung von -aB das beste Ergebnis. Der Unterschied zu dem ursprünglichen Klassifikator betrug bei -aB -0,01% und bei -R -aB 0,17%, womit -R -aB die einzige Parameterkonfiguration für Random Forest war, bei welcher der Durchschnitt positiv ausfiel. Mit -aB erzielten sechs Datensätze eine Verbesserung, mit der höchsten auf glass mit 2,34%. Mit -R -aB wurde dies ebenfalls für sechs Datensätze erreicht, zusätzlich war die Differenz bei vier weiteren Datensätzen Null. Das beste Ergebnis für -R -aB erzielte glass mit 3,74%, für -R -aB ebenfalls glass mit 3,74%. Auch wenn nur genauso viele Datensätze wie bei -R eine Verbesserung erzielten, ist das Ergebnis insgesamt besser zu bewerten, aufgrund der erreichten niedrigeren negativen Werte für die übrigen Datensätze. Der niedrigste Wert betrug für beide Parameter bei vehicel -1,18%. Für Random Forest hat sich die Nutzung beider Parameter als zweckmäßig erwiesen, da die Teilmengen der Datensätze, welche Verbesserungen erzielten, nicht identisch waren. Die signifikantesten Verbesserungen wurden mit -R -aB erreicht, damit ist dieser Parameter der durchschnittlich stärkste für Random Forest.

Mit der Nutzung von -aB bei JRip sank wie bei J48 die Durchschnittsgenauigkeit: -aB erzielte eine Verbesserung von nur 1,19% und -R -aB von 1,03%. Im Gegensatz zu J48 stieg allerdings mit -aB die Zahl der Datensätze, bei denen eine Verbesserung beobachtet werden konnte, auf zwölf, vier erzielten eine Null und nur bei solar-flare-c war eine Verschlechterung mit -0,29% zu verzeichnen. Dies ist das beste Ergebnis in Bezug auf die geringste Anzahl von Verschlechterungen aus allen Parameter- und Klassifikator-Kombinationen. -R -aB erzielte mit vierzehn Verbesserungen die höchste Anzahl in allen Tests, nur drei Datensätze zeigten eine Verschlechterung. Um eine konsistente Steigerung der Genauigkeit zu erreichen, empfehlen sich somit sowohl -aB als auch -R -aB. Ohne -aB war die Konsistenz von Verbesserungen geringer, dafür wiesen die Tests im Schnitt eine größere Verbesserung auf.

Analog zum Vergleich von -A und -R in Bezug auf Datensätze und mögliche Gemeinsamkeiten über Klassifikatoren hinweg in Kapitel 4.3 wurde dieser auch mit dem zusätzlichen Parameter -aB durchgeführt unter Verwendung derselben Kategorien. Allgemein war die Differenz zwischen den Werten der absoluten und relativen Variante geringer als im vorherigen Vergleich. Dies kann dadurch erklärt werden, dass die Einschränkungen bei der Erzeugung der Bäume wesentlich größer waren und die Bäume eine ähnlichere Form besaßen.

Solar-flare-c, Solar-flare-m, thyroid_hypo und landsat befanden sich immer in der Kategorie “-A besser“ oder “<0,1%“, soybeans, Page-blocks, optdigit in der Kategorie “-R besser“ oder “<0,1%“, thyroid_hyper, thyroid_rep, vehicel in der Kategorie “<0,1%“ und letter in der Kategorie “-A besser“. Solar-flare-c, Solar-flare-m, thyroid_hypo, Page-blocks und optdigit waren vier mal in der Kategorie “<0,1%“ vertreten.

Solar-flare-c, Solar-flare-m, thyroid_hypo, thyroid_hyper, thyroid_rep, vehicel fanden sich am häufigsten in der Kategorie “<0,1%“, aus den im vorherigen Kapitel genannten Gründen. soybeans befand sich nur dreimal in der Kategorie “<0,1%“, aber die höchste Abweichung betrug -0,29%. Damit fällt soybeans in dieselbe Kategorie wie solar-flare-m im vorherigen Vergleich. Der interessanteste Datensatz hier ist letter, bei dem -aB bei allen Klassifikatoren ein besseres

Ergebnis lieferte als das relative Gegenstück. Ohne -aB war ein solches Verhalten nicht zu beobachten. Vorwegnehmend ist zu sagen, dass die besten Ergebnisse der einzelnen Klassifikatoren über alle Parameter hinweg für letter immer mit den absoluten Varianten erreicht wurden. Da der Datensatz komplex ist und sich bei hier getesteten vergleichbaren Datensätzen keine ähnlichen Trends abzeichneten, wird angenommen, dass es sich um eine Eigenschaft des Datensatzes handelt. Dies sollte allerdings mit weiteren Tests überprüft werden.

4.5 Balance mit Parameter (-B x)

Dieses Kapitel stellt den Parameter -B vor und diskutiert die Ergebnisse seiner Anwendung. Wie in Kapitel 3.1 erläutert, fließt mit Nutzung dieses Parameters bei der Wahl der zwei zu fusionierenden Klassen zusätzlich zur Konfusionsmatrix auch die Tiefe der Subbäume ein, welche durch diese Knoten induziert werden. Der Parameter x muss eine positive reelle Zahl sein. Es wurden Testläufe mit drei verschiedenen Werten für x (0.1 / 1.0 / 10.0) durchgeführt. Mit dem ersten sollte untersucht werden, ob schon ein sehr kleiner Wert Unterschiede zum Standardfall bewirkt und wenn ja, wie diese aussehen. Der zweite ist so groß, dass er nicht vernachlässigt werden kann, sich aber noch in der gleichen Magnitude befindet. Der dritte Wert ist so gewählt, dass eine gute Balance erzwungen wird, jedoch in besonders starken Fällen der ursprüngliche Teil der Gleichung noch den größeren Einfluss hat.

Insgesamt werden in diesem Kapitel die Ergebnisse von sechs Testläufen, jeder der drei Werte mit und ohne -R, vorgestellt. Die Tabellen mit den Ergebnissen und Unterschieden zu den jeweiligen ursprünglichen Klassifikatoren befinden sich in Anhang A.1.4, die Visualisierungen der Unterschiede ebenfalls.

Unter dem Einfluss des neuen Parameters werden die Ergebnisse für Naive Bayes nur selten besser. Für -B 0,1 betrug die Durchschnittsgenauigkeit -4,23%, nur zwei Datensätze zeigten eine minimale Verbesserung. Mit -B 1,0 steigt der Durchschnitt auf -3,89%, mit wiederum den einzigen Verbesserungen auf denselben beiden Datensätzen wie bei -A. Bei Erhöhung des Parameters auf -B 10,0 sank der Durchschnitt auf -4,85% und alle Datensätze wiesen eine Verschlechterung auf. Mit Nutzung von -R waren die Ergebnisse wenig besser. -R -B 0,1 erzielte -2,99% mit substantiellen Verbesserungen für zwei Datensätze, denselben wie bei -R. Bei Erhöhung des Parameters sank die Durchschnittsgenauigkeit für -R -B 1,0 auf -3,06%. Auffallend war, dass wieder die gleichen Datensätze eine Verbesserung erzielten, wobei einer den Wert hielt und beim anderen, Page-blocks, das Ergebnis von 1,55% auf 1,85% anstieg. Dieser Trend setzte sich bei -R -B 10,0 fort. Der Durchschnitt fiel weiter auf -3,75% und Page-blocks stieg auf 2,25%. Dies ist die höchste Verbesserung, die ein Datensatz mit Naive Bayes in allen Tests erzielte. Generell sinkt die Genauigkeit für Naive Bayes mit Erhöhung des Balance Parameters, allerdings gibt es mit -R einige wenige Datensätze, auf denen die Anwendung von CT sinnvoll ist und die Genauigkeit mit höherer Balance steigt.

Ähnlich zu Naive Bayes sank für J48 mit Einführung des neuen Parameters die Durchschnittsgenauigkeit ebenfalls. Mit -B 0,1 lag der Durchschnitt bei 0,35% und damit unter dem von -A. Acht der Datensätze wiesen eine Verbesserung auf. Die Erhöhung des Parameters auf -B 1,0 veränderte den Durchschnitt nicht, allerdings verringerte sich die Anzahl der Datensätze mit Verbesserung auf sieben. Mit -B 10,0 fiel der Durchschnitt auf den negativen Wert von -0,06%,

wobei es wieder acht Datensätze mit Verbesserung gab. Die relative Variante verhielt sich ähnlich, für -R -B 0,1 lag der Durchschnitt bei 0,38% und damit geringfügig über -R. Es gab 10 Datensätze mit Verbesserung. Die Erhöhung des Parameters auf -B 1,0 verringerte die Durchschnittsgenauigkeit auf 0,21% und die Anzahl der Datensätze mit Verbesserung auf neun. Bei weiterer Erhöhung auf -R -B 10,0 fiel der Durchschnitt auf 0,03%, mit unveränderter Anzahl von Verbesserungen. Wie bei Naive Bayes ist auch bei J48 primär der konkrete Datensatz dafür verantwortlich, ob bei wachsendem Parameter das Ergebnis steigt oder fällt. Auch wenn die Durchschnittsgenauigkeit mit Verwendung von -B sinkt, gibt es Datensätze, die genauer werden mit einem hohen Parameter. Um das bestmögliche Ergebnis zu extrahieren, muss man für jeden Datensatz den optimalen Parameter suchen.

Für SMO brachte die Verwendung von -B für bestimmte Werte von x eine Verbesserung. -B 0,1 ergab eine Durchschnittsgenauigkeit von -1,19% mit sechs Datensätzen, welche eine Verbesserung aufwiesen, womit eine leichte Verbesserung erzielt wurde gegenüber der Variante ohne -B 0,1. Mit -B 1,0 stieg der Durchschnitt auf -0,79% mit verbesserten Werten für ebenfalls sechs Datensätze. Für -B 10,0 sank der Durchschnitt um über ein Prozent auf -1,98%, es waren allerdings sieben Datensätze mit Verbesserung zu verzeichnen. Die relative Variante wies ebenfalls Verbesserungen auf. -R -B 0,1 erzielte eine Durchschnittsgenauigkeit von -0,15% und acht Datensätze mit Verbesserung. -R -B 1,0 war nur minimal schwächer mit einem Durchschnitt von -0,16% und ebenfalls acht Datensätzen. Wie bei der absoluten Variante verschlechterte sich das Ergebnis bei Nutzung des größten Parameters. Für -R -B 10,0 fiel der Durchschnitt auf -1,72% mit nur fünf Datensätzen mit einer Verbesserung. Wie bei den beiden zuvor betrachteten Klassifikatoren reagierten die Datensätze sehr unterschiedlich auf die verschiedenen Werte für x. Auch hier gab es Datensätze, welche die höchste Verbesserung mit -B 10,0 erzielten, z.B. segment-challenge.

Random Forest war der einzige Klassifikator, bei dem die durchschnittliche Genauigkeit mit wachsendem x stieg, zumindest in der absoluten Variante. -B 0,1 erzielte einen Durchschnitt von -0,27% und fünf Datensätze mit einer Verbesserung, für -B 1,0 stieg der Durchschnitt auf -0,24% mit ebenfalls fünf Datensätzen. Bei -B 10,0 waren es ebenfalls fünf Datensätze, der Durchschnitt stieg weiter auf -0,13%. Für die relative Variante ist dieser Trend nicht zu verzeichnen. Der Durchschnitt für -R -B 0,1 lag bei -0,06% und es wurden sechs Datensätze mit Verbesserung beobachtet. Für -R -B 1,0 stieg der Durchschnitt auf -0,03%, gleichzeitig fiel allerdings die Zahl der Datensätze mit Verbesserung auf fünf. Mit -R -B 10,0 sank der Durchschnitt leicht auf -0,05%, jedoch wuchs die Zahl der Datensätze auf sieben an. Wie bei den vorhergehenden Klassifikatoren gab es einzelne Datensätze, die von einer hohen Balance profitierten, und andere, für welche dies gegenläufig war. Bei Random Forest traf dies zum Beispiel auf den Datensatz landsat zu, welcher mit -R -B 10,0 das beste Ergebnis erzielte.

JRip kehrte wieder zum Trend der anderen Klassifikatoren zurück, mit höherem x sank die Durchschnittsgenauigkeit. -B 0,1 startete mit einem Durchschnitt von 1,31%, leicht schlechter als -A, mit -B 1,0 sank der Durchschnitt auf 1,27%, mit -B 10,0 auf 0,87%. Für alle drei Werte gab es gleichbleibend jeweils zwölf Datensätze mit einer Verbesserung. Bezüglich der Durchschnittsgenauigkeit folgte die relative Variante demselben Abwärtstrend. -R -B 0,1 begann mit einem Durchschnitt von 1,18%, fiel leicht auf 1,17% mit Erhöhung von x auf 1,0 und sank mit -R -B 10,0 weiter auf 0,97%. Während für die ersten beiden Werte von x jeweils Verbesserungen für zehn Datensätze zu verzeichnen waren, stieg die Anzahl für den dritten Wert auf elf. vehicel und segment-challenge erreichten für JRip ihre besten Ergebnisse mit einem x von

10,0. Abbildung 10 zeigt die relative Variante. Die einzelnen Datensätze sind durchnummeriert in derselben Reihenfolge wie in der Auflistung in Kapitel 4.1, dementsprechend sind segment-challenge und vehicel durch die Nummern 1 und 13 repräsentiert. Ein Anstieg der Genauigkeit mit wachsendem x ist jeweils gut sichtbar.

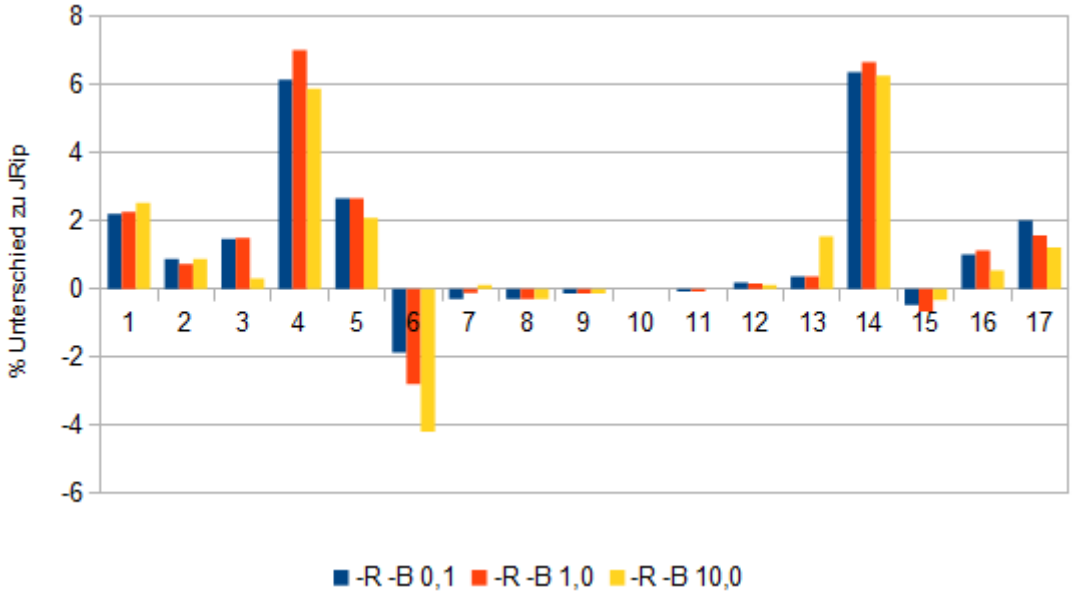


Abbildung 10: Genauigkeitsunterschiede von JRip zu CT -R -B 0.1 , 1.0 und 10.0 mit JRip

Wie in den vorhergehenden Kapiteln wurden auch für -B x die einzelnen Datensätze dahingehend untersucht, ob Gemeinsamkeiten über Klassifikatoren hinweg auftraten. Auffallend war der Datensatz car. Beispielsweise für J48 stieg die Genauigkeit von -0,69% über -0,46% auf 0,12% in der absoluten Variante. In der relativen fiel sie stattdessen von 1,50% über 0,52% auf ebenfalls 0,12%. Dieser Trend der steigenden Genauigkeit mit größerem x bei -A und fallender Genauigkeit bei -R war für car bei allen Klassifikatoren zu beobachten. Im direkten Vergleich der absoluten und relativen Variante eines Parameters verzeichnete -R immer die besseren Ergebnisse, mit einer Ausnahme bei SMO mit -B 10,0. Kein anderer Datensatz wies über alle Klassifikatoren hinweg eine solche strukturierte Tendenz auf. Das kann für car damit erklärt werden, dass dieser Datensatz über eine sehr geringe Anzahl von Klassen und Attributen verfügt. Mit steigender Komplexität der getesteten Datensätze konnten solche Merkmale nur in immer schwächerer Form beobachtet werden.

Andere Datensätze wiesen die folgenden Tendenzen auf: bei segment-challenge brachte ein größeres x bei vier von fünf Klassifikatoren eine größere oder zumindest gleich große Genauigkeit mit sich. Bei optdigit sank die Genauigkeit mit -R mit größerem x für alle getesteten Klassifikatoren. Entsprechend traf dies für letter und vowel zu, jedoch nur für vier der fünf Klassifikatoren. Die Ergebnisse für thyroid_hypo blieben für vier Klassifikatoren konstant, unabhängig von den verschiedenen Werten von x, wobei die absolute und relative Variante unterschiedliche Konstanten aufwiesen. Für Solar-flare-c, Solar-flare-m, thyroid_hyper, thyroid_rep, vehicel traf dies auf drei Klassifikatoren zu.

4.6 Baumstruktur

Ein weiterer untersuchter Aspekt waren die von CT erzeugten Baumstrukturen, wie diese sich bei verschiedenen Parametern unterscheiden und ob bestimmte Strukturen für bestimmte Datensätze oder Klassifikatoren besser geeignet sind. In diesem Kapitel werden die in den Tests beobachteten Baumstrukturen vorgestellt und mit den in Kapitel 3.1 beschriebenen, erwarteten Baumstrukturen der einzelnen Parameter verglichen.

Für -A bildeten sich nicht, wie in Kapitel 3.1 angenommen, mehrere Ketten, sondern nur eine einzelne lange. Häufiger bildeten sich am unteren Ende der Kette kurze Abzweigungen, in Abhängigkeit von der Anzahl der Klassen im Datensatz oft nur ein oder zwei Knoten lang. Es ist anzunehmen, dass diese Abzweigungen mit steigender Anzahl von Klassen in den Datensätzen länger werden.

Abbildung 11 zeigt die typische Struktur eines Baumes, wie sie bei Nutzung von -A entstanden ist. In dieser und allen folgenden Abbildungen von Bäumen repräsentieren die Nummern in den Blättern die einzelnen Klassen. Die zwei durch einen vertikalen Strich getrennten Nummern in den inneren Knoten stellen dar, welche zwei Klassen in ihnen zusammengefasst sind. Diese Metaklasse wird weiter oben im Baum immer durch die kleinere der beiden Klassennummern repräsentiert.

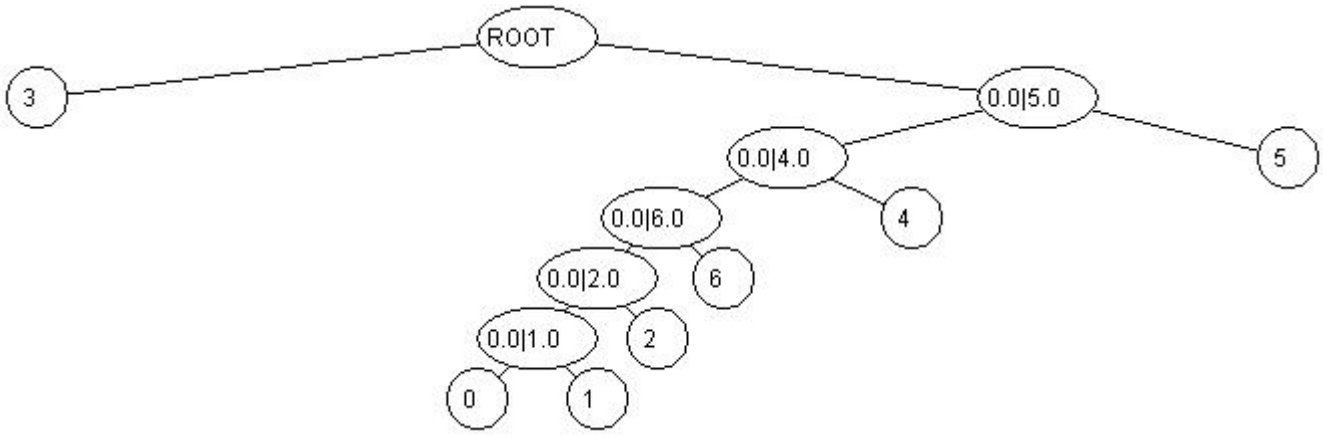


Abbildung 11: Baum des Datensatzes glass in der Standardvariante von CT mit J48

Für -R wurden die in Kapitel 3.1 getroffenen Aussagen durch die Tests bestätigt. Die beobachtete Baumstruktur war sehr zufällig. Es wurden Fälle mit Baumstrukturen in Form einer einzelnen langen Kette, ähnlich zu -A, dokumentiert. In anderen war der Baum nahezu ausbalanciert. Viele zwischen diesen beiden Extremen liegende Baumstrukturen konnten ebenso beobachtet werden. Eine Auswahl dieser ist in Anhang A.2 aufgeführt.

Für -aB trafen die beschriebenen Annahmen zu, allerdings waren frühe Abbrüche eher die Regel als die Ausnahme, womit es häufig Gruppen von Klassen geben muss, die keine Überlappung besitzen. Die relative und absolute Variante von -aB bildeten dieselbe Struktur von Bäumen, auch wenn sich die einzelnen Metaklassen und damit das Ergebnis mitunter stark unterschieden. Dabei waren Abbrüche in der absoluten Variante etwas häufiger zu verzeichnen. Abbildung 12 stellt einen typischen Baum für -aB dar.

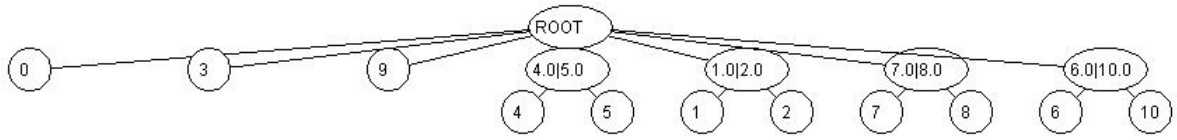


Abbildung 12: Baum des Datensatzes vowel von CT -aB mit SMO

Die Baumstruktur für -B in den Tests entsprach den in Kapitel 3.1 beschriebenen Erwartungen: je größer x, desto flacher und balancierter war der Baum. Für den Parameter 0,1 waren die Bäume nahezu identisch mit denen ohne -B. Erst bei Datensätzen mit der größten Anzahl von Klassen, die in den Tests verwendet wurden, traten kleinere Differenzen auf. Häufig war die Tiefe des Baumes in diesen Fällen um eins oder zwei kleiner, selten um größere Beträge. Im Vergleich von -B 1,0 und -B 0,1 sehen die Baumstrukturen für Datensätze mit wenigen Klassen in der Regel identisch aus, damit wie ohne -B. Erst mit einer höheren Anzahl von Klassen waren regelmäßig Unterschiede zu beobachten. Zum Beispiel für letter sank die Tiefe um fünf im Durchschnitt über alle Klassifikatoren bei einem Wechsel von -B 0,1 auf -B 1,0. Bei glass waren die Bäume für alle Klassifikatoren für beide Parameter identisch. Für -B 10,0 erreichten die meisten Bäume die minimal mögliche Tiefe, allerdings nicht unbedingt eine perfekte Balance. Eine unterschiedliche Anzahl von Klassen in den Datensätzen hatte keine Auswirkungen auf dieses Ergebnis. In Anhang A.2 finden sich beispielhaft die Abbildungen 38, 39 und 40 der entstandenen Bäume für den Datensatz vowel für alle drei Werte des Parameters.

Bei zusätzlicher Verwendung von -R wird der Baum für kleinere x deutlich flacher. Dies relativiert sich mit größeren Werten für x. Für -R -B 10,0 war die Tiefe nahezu identisch im Vergleich zu -B 10,0, mit Ausnahme weniger Fälle, bei denen im absoluten Fall die Tiefe des Baumes um eins größer war.

Wenn die relative und absolute Variante dieselbe Genauigkeit erzielen, ist die Benutzung der relativen zu empfehlen, da durch den flacheren Baum kürzere Zeiten beim Klassifizieren erreicht werden.

4.7 Zusammenfassung

In diesem Abschnitt werden die Ergebnisse der experimentellen Evaluierung zusammengefasst und ein Fazit über die Effektivität einzelner Parameter und die Stärke von CT auf den jeweiligen Klassifikatoren gezogen. Begonnen wird mit einem Beispiel, welches aufzeigt, dass CT in der Praxis sich so verhält wie beabsichtigt.

In der nachfolgenden Abbildung 13 wird am Beispiel des Datensatzes letter mit J48 und -B 10,0 visualisiert, dass CT die gewünschte Funktionsweise gezeigt hat. Für letter müssen Buchstaben klassifiziert werden. Dementsprechend hat der Datensatz 26 Klassen, eine für jeden Buchstaben des Alphabets. In der Abbildung entspricht die Klasse null einem a, eins einem b, zwei einem c und so fort. Das Beispiel Buchstaben wurde gewählt, da intuitiv nachvollziehbar ist, welche in Metaklassen zusammengefasst werden sollten, hier zum Beispiel in der untersten Reihe O und Q (14 und 16), I und J (8 und 9)oder W und M (12 und 22). Damit wurden wie beabsichtigt

ähnliche Klassen in Metaknoten zusammengefasst, um sie später unter Abwesenheit störender Einflüsse besser trennen zu können.

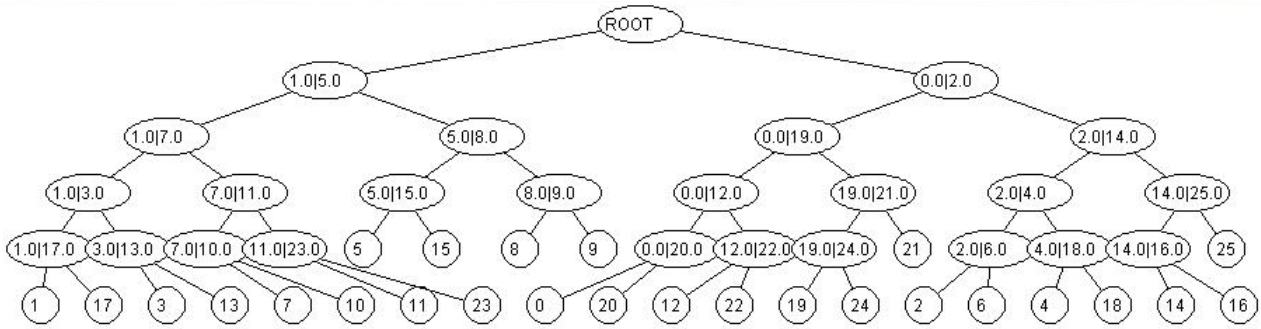


Abbildung 13: Baum des Datensatzes letter von CT -B 10,0 mit J48

In den Unterkapiteln 4.2 bis 4.5 wurden die Ergebnisse der Tests mit den einzelnen Parametern vorgestellt. Eine verbleibende Frage ist, für welchen Datensatz und Klassifikator welcher Parameter das beste Ergebnisse lieferte. Tabelle 7 listet für jeden Datensatz und Klassifikator diese Parameter auf. Zu beachten ist, dass dies jeweils das beste mit CT erzielte Ergebnis und nicht unbedingt besser als der ursprüngliche Klassifikator war.

Wie in Tabelle 7 zu sehen, haben die besten Parameter für die meisten Datensätze kein erkennbares Muster. “all“ steht dafür, dass dasselbe Ergebnis auf allen Parametern erzielt wurde. Ausnahmen waren die Datensätze solar-flare-c, welcher immer mit -B 0,1 und -B 1,0 das beste Ergebnis erzielte, und thyroid_hypo, wo dies für -aB immer zutraf. Zusätzlich zu den genannten Parametern gab es weitere für die verschiedenen Klassifikatoren, mit denen ebenfalls das beste Ergebnis erreicht wurde. Weiterhin gehören die genannten zu den insgesamt drei Datensätzen, die mit mindestens einem Klassifikator für alle Parameter dasselbe Ergebnis erzielten. Wie in vorangegangenen Kapiteln beschrieben, sind diese in die Kategorie der einfach zu klassifizierenden Datensätze einzuordnen und daher ist anzunehmen, dass diese Muster auf die dort genannten Gründe zurückzuführen sind.

In der Gesamtbetrachtung der Ergebnisse traten außerdem die beiden Datensätze letter und optdigit hervor. letter erzielte das beste Ergebnis ausschließlich für -A oder -aB, optdigit für -R -B 0,1 oder -R -aB. Beide gehörten zu den komplexeren der getesteten Datensätze, daher war dieser Trend umso erstaunlicher. Dass es sich hier tatsächlich um ein charakteristisches Verhalten dieser Datensätze handelt, wäre mit weiteren Tests mit anderen Klassifikatoren zu untersuchen. Dieser Nachweis konnte im Rahmen dieser Arbeit nicht erbracht werden.

Es konnte aufgezeigt werden, dass bestimmte Datensätze regelmäßig auf bestimmte Parameter ansprechen, andere wiederum keine Präferenzen zeigten. Um eine verlässliche Aussage pro Datensatz treffen zu können, sollte für jeden eine höhere Anzahl von Tests mit jeweils mehr als fünf Klassifikatoren durchgeführt werden. Aus Ähnlichkeiten zwischen den Datensätzen bezüglich Anzahl von Klassen, Instanzen oder Attributen, wie zum Beispiel für die hier getesteten Datensätze thyroid_hyper, thyroid_hypo und thyroid_rep, kann nicht auf ein ähnliches Verhalten bezüglich der Parameter geschlossen werden, dieses ist für jeden Datensatz spezifisch zu ermitteln.

Tabelle 7: Diese Tabelle listet für jeden Datensatz und Klassifikator die Parameter, welche das beste Ergebnis erzielt haben.

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-A, -R, -B 0.1, -B 1, -R -B 0.1, -R -B 1	-R -aB	-R -B 10, -B 10	-A, -R, -B 0.1, -B 1, -B 10 , -R -B 0.1, -R -B 1, -R -B 10	-R -B 10
soybeans	-aB, -R -aB	-B 10, -R -B 10	-B 0.1 ,-B 1	-R, -R -aB, -R -B 0.1, -R -B 1, -R -B 10	-B 0.1, -B 1
letter	-aB	-A	-aB	-aB	-A
abalone	-R	-R 1	-aB	-R -B 10	-R -B 1
car	-R, -R -B 0.1, -R -B 1, -R -B 10	-R, -R -B 0.1	-aB, -B 10	-R -aB	-R, -R -B 0.1, -R -B 1
glass	-aB	-A	-R -aB, -R -B 1 ,-R -B 10	-R -aB	-A
Page-blocks	-R -aB, -R -B 10	-R -aB	-A, -B 0.1, -B 1, -B 10	-R -aB	-aB
Solar-flare-c	-A, -B 0.1, -B 1	-A, -aB, -B 0.1, -B 1, -B 10	all	-R, -R -B 0.1, -R -B 1	all
Solar-flare-m	-aB, -R -aB	all	all	-A, -aB, -B 0.1, -B 1, -B 10	-R -aB
thyroid_hyper	-A, -B 0.1, -B 1	-R, -R -aB, -R -B 0.1, -R -B 1, -R -B 10	-R, -R -B 0.1, -R -B 1, -R -B 10	-R, -R -B 0.1, -R -B 1, -R -B 10	-B 10
thyroid_hypo	-A, -aB, -B 0.1, -B 1.0, -B 10	-A, -aB, -B 0.1, -B 1, -B 10	all	-aB	-aB, -R -aB
thyroid_rep	-R, -R -B 0.1 ,-R -B 1	-R, -R -aB, -R -B 0.1, -R -B 1, -R -B 10	-aB, -R -aB	-aB	-A
vehicel	-R, -aB, -R -aB, -B 1, -B 10, -R -B 0.1, -R -B 1, -R -B 10	-aB, -R -aB, -R -B 10, -B 10	-A, -R, -B 0.1, -B 1, -R -B 0.1, -R -B 1	-A, -R, -B 0.1, -B 1, -B 10, -R -B 0.1, -R -B 1, -R -B 10	-B 10, -R -B 10
vowel	-R	-R -aB	-R, -R -B 0.1, -B 0.1	-R -aB	-A, -R -aB
yeast	-A, -B 0.1	-R -B 1	-R -aB	-B 10	-R -aB
landsat	-R, -B 1, -R -B 0.1, -R -B 1	-B 10	-R -B 10	-R -B 10	-A, -B 0.1
optdigit	-R -B 0.1	-R -B 0.1	-R -aB	-R -aB	-R -B 0.1

Im Gegensatz zu den Datensätzen ist für die Klassifikatoren deutlich geworden, dass es durchaus Parameter gibt, die eine höhere Performance aufweisen als andere. Für Naive Bayes war dies -aB, für J48 -A, für SMO -R -B 0.1, für Random Forest -R -aB und für JRip -A. Diese erzielten im Durchschnitt jeweils die besten Ergebnisse. Wie weiterhin deutlich geworden und in Tabelle 7 erkennbar ist, erzielten für spezifische Datensätze häufig unterschiedliche Parameter das beste Ergebnis. Was in Tabelle 7 nicht deutlich wird ist, ob das beste Ergebnis eine Verbesserung zum ursprünglichen Klassifikator darstellt. Dies zeigt Tabelle 8 auf. Ein "+" repräsentiert, dass die Parameter mit dem besten Ergebnis im Vergleich zum ursprünglichen Klassifikator eine Verbesserung erzielten, eine "0" ein unverändertes Ergebnis und ein "-", dass keine Verbesserung erreicht werden konnte.

Tabelle 8: Diese Tabelle listet für jeden Datensatz und Klassifikator auf, ob es einen Parameter gab, der ein besseres Ergebnis erzielte als der ursprüngliche Klassifikator.

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	-	+	+	+	+
soybeans	-	+	-	-	+
letter	-	-	-	-	+
abalone	-	+	+	+	+
car	+	+	-	+	+
glass	-	+	+	+	+
Page-blocks	+	+	+	+	+
Solar-flare-c	+	+	0	-	-
Solar-flare-m	-	0	0	+	0
thyroid_hyper	+	-	+	+	+
thyroid_hypo	-	-	+	-	+
thyroid_rep	0	+	0	+	+
vehicel	-	-	-	-	+
vowel	-	-	+	+	+
yeast	-	+	-	-	+
landsat	-	-	+	+	+
optdigit	-	+	-	0	+

Wie in den Kapiteln 4.2 bis 4.5 bereits erläutert wurde, ist die Performance von Naive Bayes mit CT unbefriedigend. Bei nur vier von siebzehn Datensätzen zeigte sich eine Verbesserung und nur ein weiterer erreichte eine Null. SMO erzielte mit acht Verbesserungen, drei Nullen und sechs Verschlechterungen ein zufriedenstellendes Ergebnis, wobei bei keinem Parameter eine positive Durchschnittsgenauigkeit erreicht wurde. J48 und Random Forest verzeichneten ein geringfügig besseres Ergebnis. Beide erzielten jeweils bei zehn Datensätzen eine Verbesserung, bei einem eine Null und den anderen sechs eine Verschlechterung. Dabei war für J48 bis auf einen Parameter der Durchschnitt immer positiv, bei Random Forest gilt dies umgekehrt: nur ein einziger Parameter erreichte einen positiven Durchschnitt. Dies zeigt deutlich, dass Durchschnittsgenauigkeiten mit Vorsicht zu betrachten sind und im Allgemeinen kein Parameter einen anderen dominiert, jeder weist eine unterschiedliche Effizienz auf unterschiedlichen Datensätzen und Klassifikato-

ren auf. Deshalb sind, um das jeweils bestmögliche Ergebnis mit CT zu erzielen, Tests mit allen Parametern notwendig, auch wenn der dafür einzusetzende Zeitaufwand erheblich sein kann.

Unangefochtener Sieger in den Tests war JRip. Bei fünfzehn der Datensätze wurde eine Verbesserung erzielt, bei einem weiteren eine Null und für nur einen konnte keine Verbesserung erreicht werden. Selbst bei diesem lag die höchste erreichte Genauigkeit nur knapp unter der des ursprünglichen Klassifikators. Dieses Ergebnis ist wertvoll, da JRip in der Praxis bereits Anwendung findet und mit CT eine weitere Optimierung möglich ist, sofern die zusätzlichen Ressourcen für die Erzeugung des Klassifikators aufgebracht werden können. Aufgrund der guten Performance auf der den Tests zugrunde liegenden Auswahl von Datensätzen kann angenommen werden, dass CT mit JRip auch mit weiteren Datensätzen ähnlich gute Ergebnisse liefert.

5 Ausblick

In dieser Arbeit wurde der Meta-Klassifikator Classifier Trees vorgestellt. CT erzeugt aus einem Multiklassen-Klassifikator eine Baumstruktur, bei der sich in jedem inneren Knoten ein Klassifikator befindet. Die Klassifikation erfolgt ähnlich zu anderen Baum-Klassifikatoren. Es wurden Testläufe auf 17 Datensätzen, fünf Klassifikatoren und zehn verschiedenen Eingabeparametern durchgeführt, um die Effektivität von CT zu ermitteln. Für Naive Bayes funktionierte CT nur unbefriedigend, auf SMO, J48 und Random Forest zufriedenstellend und sehr gut für JRip.

Das Hauptanliegen der Arbeit bestand in der Vorstellung des Algorithmus und der Beweisführung, dass CT für einige Klassifikatoren und Datensätze eine relevante Steigerung der Genauigkeit bewirken kann. Kaum betrachtet und nicht genauer untersucht wurden die Kosten dieser Steigerung. Die Erstellungsdauer eines CT Klassifikators ist deutlich höher als die des ursprünglichen, welcher in der ersten Iteration erzeugt wird. Maßgeblich für die Erstellungsdauer von CT ist, wie sich die Erzeugungsdauer des ursprünglichen Klassifikators zur Anzahl der Klassen verhält. Ist das Verhältnis linear, so beträgt die Dauer circa $n^2/2$, wobei n die normale Erstellungsdauer des ursprünglichen Klassifikators ist. Optimierungen am Algorithmus sind möglich, insbesondere durch Parallelisierung der Doppelschleife, welche die zwei zu fusionierenden Klassen auswählt, und der Schleife, welche die Klassen der einzelnen Instanzen des Datensatzes anpasst. Die Dauer zum Klassifizieren einer Instanz hängt stark von der Baumstruktur ab. Parameter, die flachere und balancierte Bäume erzeugen, sind hier zu bevorzugen. Handelt es sich um einen balancierten Baum, sollte die Dauer nicht wesentlich größer sein als die bei dem ursprünglichen Klassifikator benötigte Dauer. Dies wurde im Rahmen dieser Arbeit nicht überprüft.

Eine weiterführende Forschung könnte mehrere Ansätze verfolgen. Während in dieser Arbeit fünf strukturell unterschiedliche Klassifikatoren getestet wurden, wäre die Untersuchung von Klassifikatoren ähnlicher Struktur denkbar, um eventuelle Muster zu finden und zum Beispiel zu klären, ob CT mit allen Regellernern so gut funktioniert wie mit JRip. Entsprechend könnte für andere Klassifikatoren überprüft werden, ob CT bei ihnen eine Verbesserung der Genauigkeit bewirkt. Weiterhin wurden die in dieser Arbeit getesteten Klassifikatoren in ihrer Grundform verwendet. SVM's wie SMO sind dafür bekannt, erst dann das bestmögliche Ergebnis für einen Datensatz zu liefern, wenn die Parameter an diesen angepasst wurden. Es wäre zu untersuchen, wie CT sich unter solchen Bedingungen verhält, da sich diese Spezifikationen auf alle von CT erzeugten Klassifikatoren auswirken würden. Ein dritten Ansatz bieten die Datensätze. Eine Untersuchung, wie sich CT auf Datensätzen mit hunderten oder tausenden Klassen verhält, würde die tatsächliche Praxistauglichkeit des Algorithmus verifizieren.

Eine weitere Vorgehensweise könnte darin bestehen, den Algorithmus selbst zu verbessern. Denkbar wäre ein neuer Parameter, der das Abbruchkriterium auflockert, sodass bereits dann abgebrochen wird, wenn der Anteil der falsch klassifizierten Instanzen im Vergleich zu den richtig klassifizierten nicht groß genug ist. Ein weiterer Parameter könnte bewirken, dass, wenn zwei Klassen gefunden sind, die Genauigkeit des gesamten Klassifikators mit und ohne den neuen Meta-Knoten ermittelt wird. Wenn sie mit dem neuen Knoten geringer ist, wird die zweitbeste Klassenkombination ausgewählt und ebenfalls diesem Test unterzogen. Nach einer definierten maximalen Anzahl von Wiederholungen wird abgebrochen, wenn bis dahin kein passendes Klassenpaar gefunden wurde.

In der Gesamtbetrachtung der Testergebnisse zeigt Classifier Trees Potential für einzelne Klassifikatoren. Um jedoch eine definitive Praxistauglichkeit nachzuweisen, sind weitere Testläufe mit anderen Klassifikatoren und Datensätzen und gegebenenfalls eine Optimierung des Algorithmus notwendig.

A Anhang

A.1 Visualisierungen der Unterschiede von CT zu den Ursprungsklassifikatoren

A.1.1 Absolute Überlappung

Tabelle 9: Genauigkeit der Standardvariante von CT auf den Testdatensätzen in Prozent

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	76,00	96,13	92,80	97,33	95,33
soybeans	83,31	92,09	93,56	90,92	93,27
letter	50,90	87,87	69,25	91,21	88,22
abalone	12,40	25,40	21,28	23,75	26,29
car	84,32	91,67	92,19	93,98	87,33
glass	38,79	70,56	56,07	75,70	69,63
Page-blocks	88,87	97,02	93,57	97,09	96,77
Solar-flare-c	80,20	85,57	85,16	84,17	85,11
Solar-flare-m	90,21	95,10	95,10	93,59	94,74
thyroid_hyper	95,41	98,83	97,88	98,70	98,54
thyroid_hypo	95,20	99,52	93,85	98,86	99,26
thyroid_rep	92,95	99,10	96,71	98,52	99,15
vehicel	40,54	70,09	74,11	75,30	69,39
vowel	58,48	80,71	74,24	94,24	76,87
yeast	57,08	58,83	51,68	57,55	57,21
landsat	76,98	85,14	87,06	89,36	86,74
optdigit	86,19	90,98	96,89	94,47	92,3

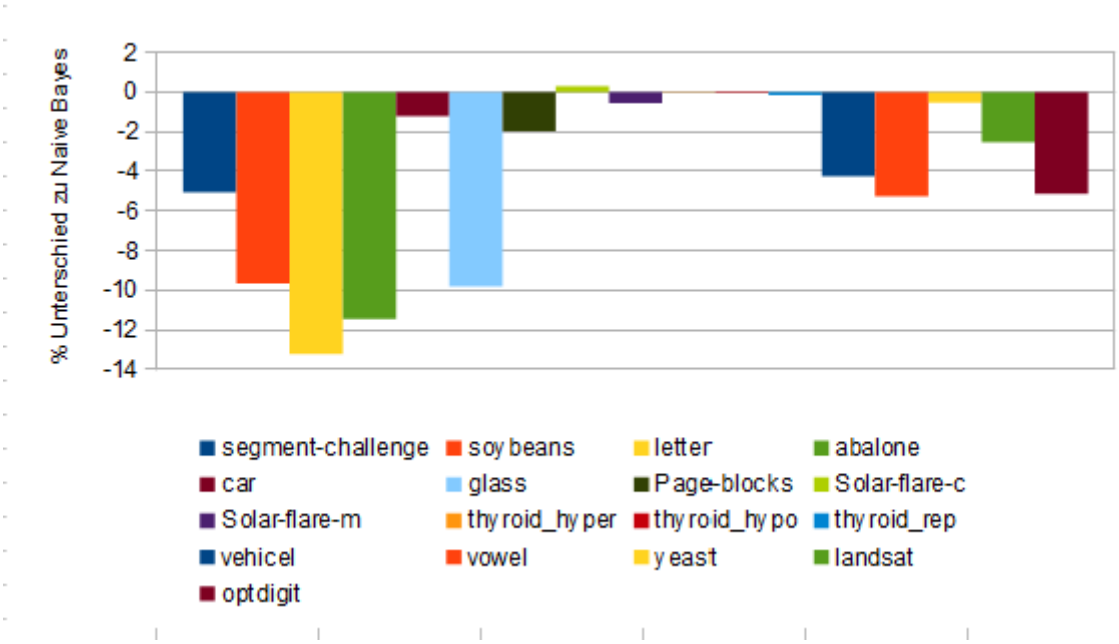


Abbildung 14: Genauigkeitsunterschiede von Naive Bayes zu CT mit Naive Bayes mit -A

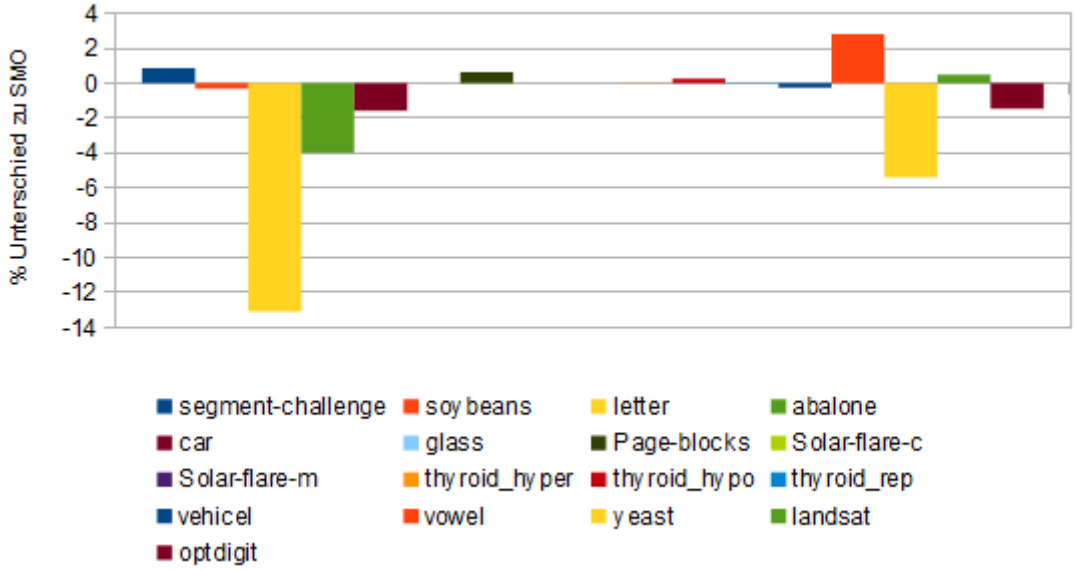


Abbildung 15: Genauigkeitsunterschiede von SMO zu CT mit SMO mit -A

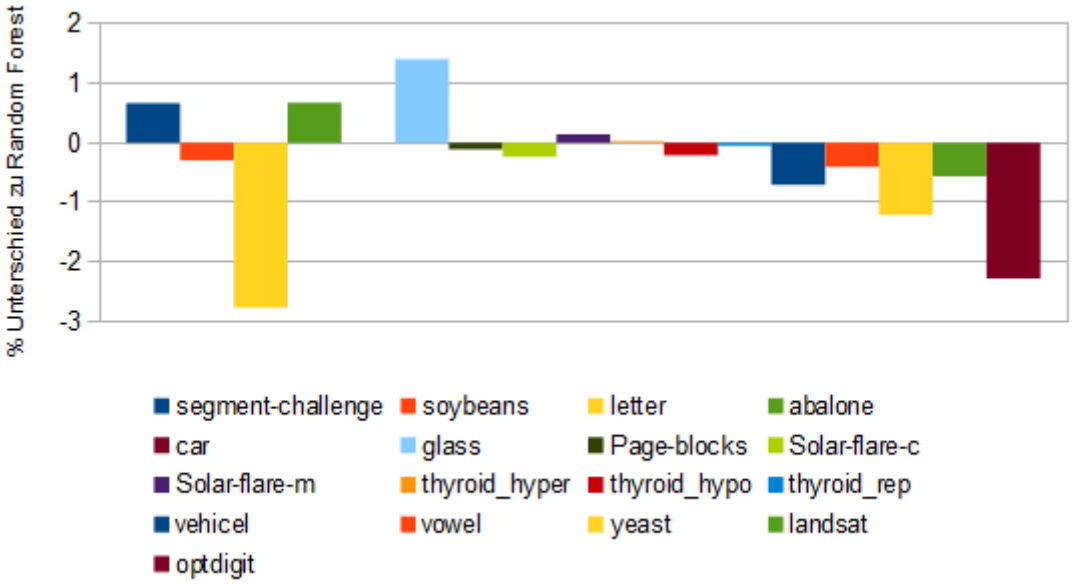


Abbildung 16: Genauigkeitsunterschiede von Random Forest zu CT mit Random Forest mit -A

A.1.2 relative Überlappung

Tabelle 10: Genauigkeit der relativen Variante (-R) von CT auf den Testdatensätzen in Prozent

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	76,00	96,00	92,80	97,33	95,33
soybeans	84,48	92,24	93,41	91,07	93,12
letter	51,40	87,41	72,61	91,53	87,70
abalone	23,51	25,26	25,45	23,46	26,10
car	87,33	93,87	93,11	94,68	89,12
glass	39,25	67,29	57,94	77,10	67,29
Page-blocks	92,40	96,99	93,02	97,22	96,55
Solar-flare-c	77,22	85,51	85,16	84,35	85,11
Solar-flare-m	90,06	95,10	95,10	93,38	94,60
thyroid_hyper	95,02	98,86	97,91	98,73	98,49
thyroid_hypo	94,67	99,50	93,85	98,94	99,26
thyroid_rep	93,11	99,18	96,71	98,52	99,13
vehicel	42,55	70,09	74,11	75,30	69,39
vowel	60,81	80,40	78,18	94,55	76,16
yeast	56,54	58,76	55,60	57,82	57,61
landsat	77,32	85,70	87,22	89,52	86,65
optdigit	86,19	91,35	96,74	95,93	92,62

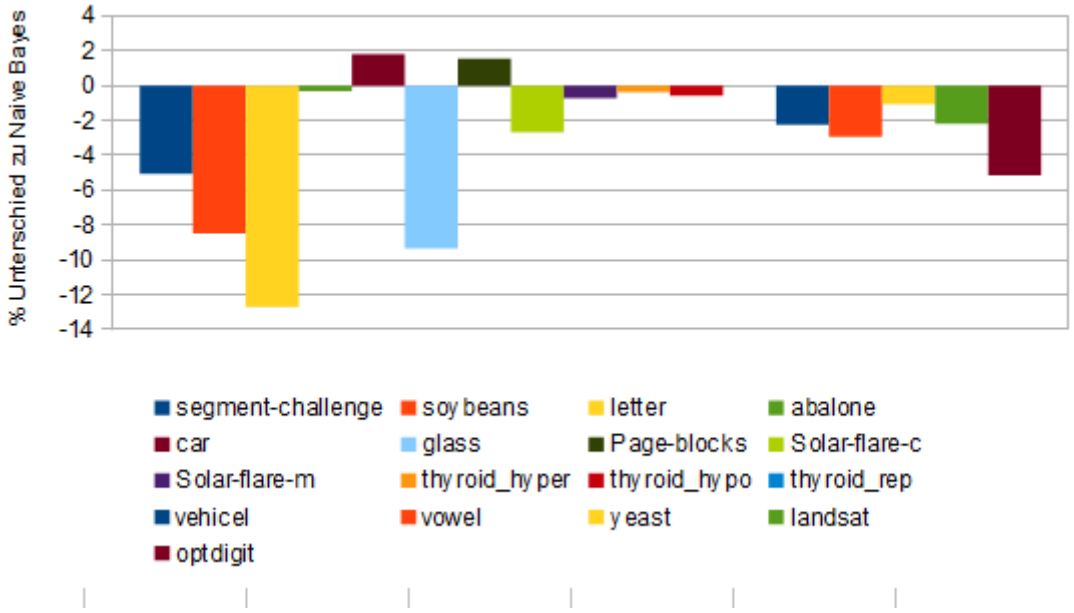


Abbildung 17: Genauigkeitsunterschiede von Naive Bayes zu CT -R mit Naive Bayes

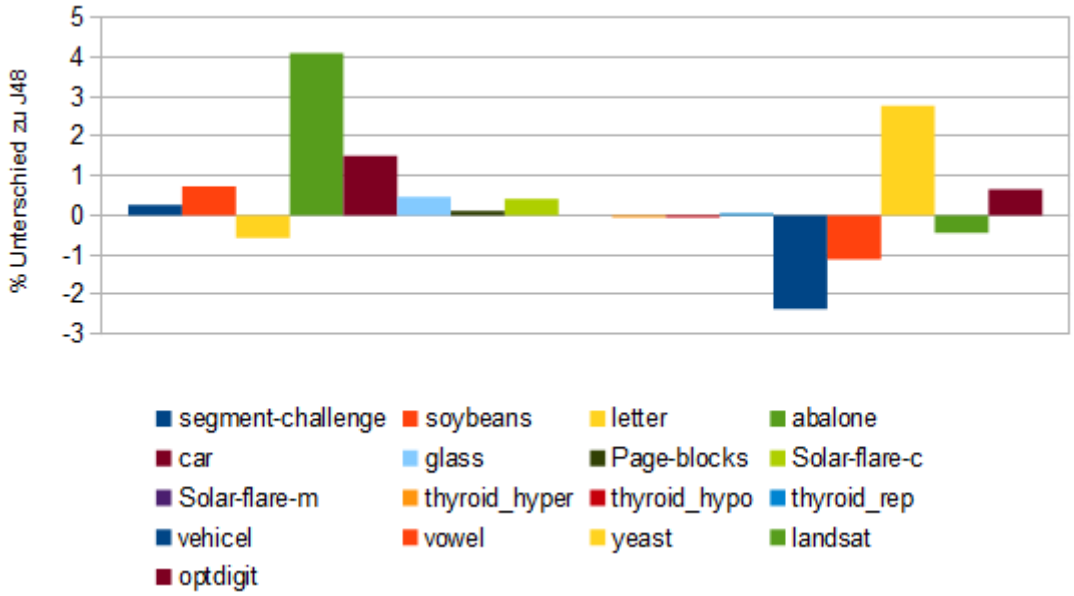


Abbildung 18: Genauigkeitsunterschiede von J48 zu CT -R mit J48

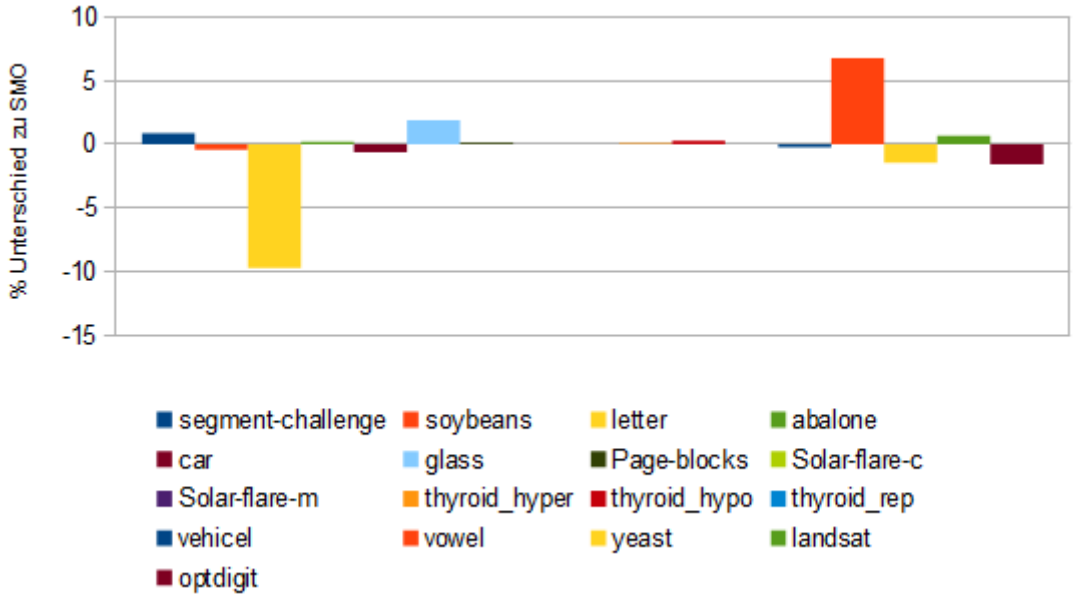


Abbildung 19: Genauigkeitsunterschiede von SMO zu CT -R mit SMO

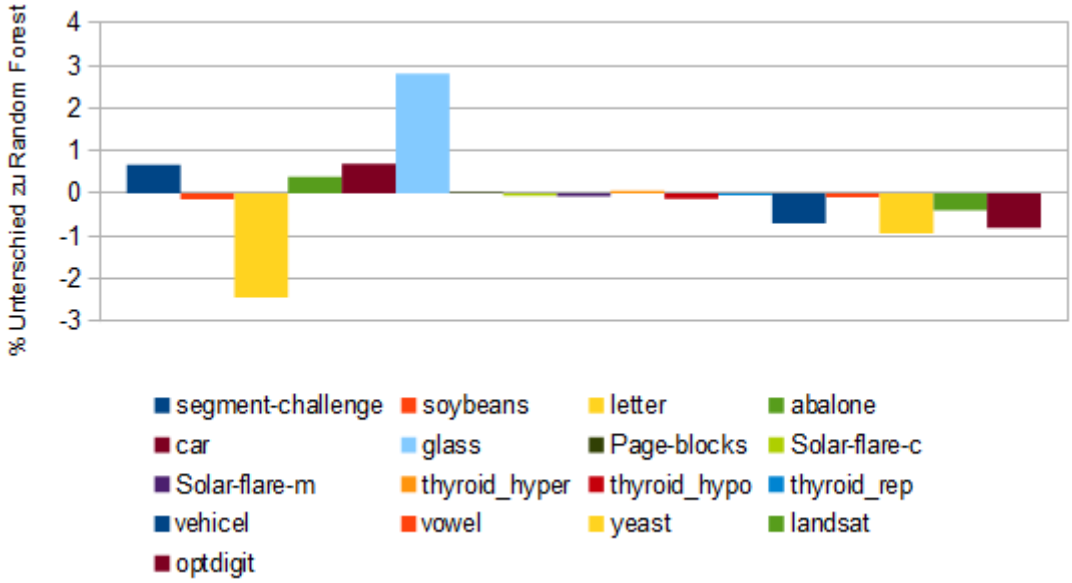


Abbildung 20: Genauigkeitsunterschiede von Random Forest zu CT -R mit Random Forest

A.1.3 absolute Balance

Tabelle 11: Genauigkeit der absoluten Variante mit absoluter Balance (-aB) von CT auf den Testdatensätzen in Prozent

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	74,20	96,07	92,73	96,87	95,60
soybeans	92,39	92,53	93,27	90,78	92,53
letter	57,96	86,71	75,27	93,73	86,71
abalone	22,98	23,49	25,54	23,87	25,64
car	85,07	92,48	93,23	93,75	87,96
glass	46,73	66,36	58,88	76,64	68,69
Page-blocks	88,80	97,04	93,51	97,28	96,97
Solar-flare-c	79,67	85,57	85,16	84,23	85,11
Solar-flare-m	90,42	95,10	95,10	93,59	94,74
thyroid_hyper	95,18	98,83	97,88	98,62	98,49
thyroid_hypo	95,20	99,52	93,85	98,97	99,39
thyroid_rep	93,00	99,10	96,74	98,62	99,05
vehicel	42,55	70,21	69,98	74,82	69,86
vowel	60,30	75,25	68,38	94,04	76,57
yeast	56,81	57,08	55,12	58,49	58,09
landsat	77,70	85,73	86,70	89,90	86,36
optdigit	81,25	89,98	97,12	96,30	91,65

Tabelle 12: Genauigkeit der relativen Variante mit absoluter Balance (-R -aB) von CT auf den Testdatensätzen in Prozent

Datensatz	NB	J48	SMO	RF	JRip
segment-challenge	74,20	96,27	92,73	96,87	95,40
soybeans	92,39	92,53	93,27	91,07	92,68
letter	53,07	86,58	74,88	93,58	86,66
abalone	23,46	23,56	24,13	23,65	25,11
car	86,75	92,48	93,11	94,79	88,54
glass	45,33	66,82	59,35	78,04	65,89
Page-blocks	93,09	97,09	93,51	97,33	96,95
Solar-flare-c	79,15	85,51	85,16	84,23	85,11
Solar-flare-m	90,42	95,10	95,10	93,45	94,82
thyroid_hyper	95,12	98,86	97,88	98,67	98,46
thyroid_hypo	94,64	99,50	93,85	98,94	99,39
thyroid_rep	93,05	99,18	96,74	98,57	99,05
vehicel	42,55	70,21	69,98	74,82	69,86
vowel	59,39	80,81	71,31	94,75	76,87
yeast	56,33	57,01	56,47	58,22	58,42
landsat	77,70	85,55	86,25	89,88	85,95
optdigit	81,25	90,34	97,17	96,74	91,67

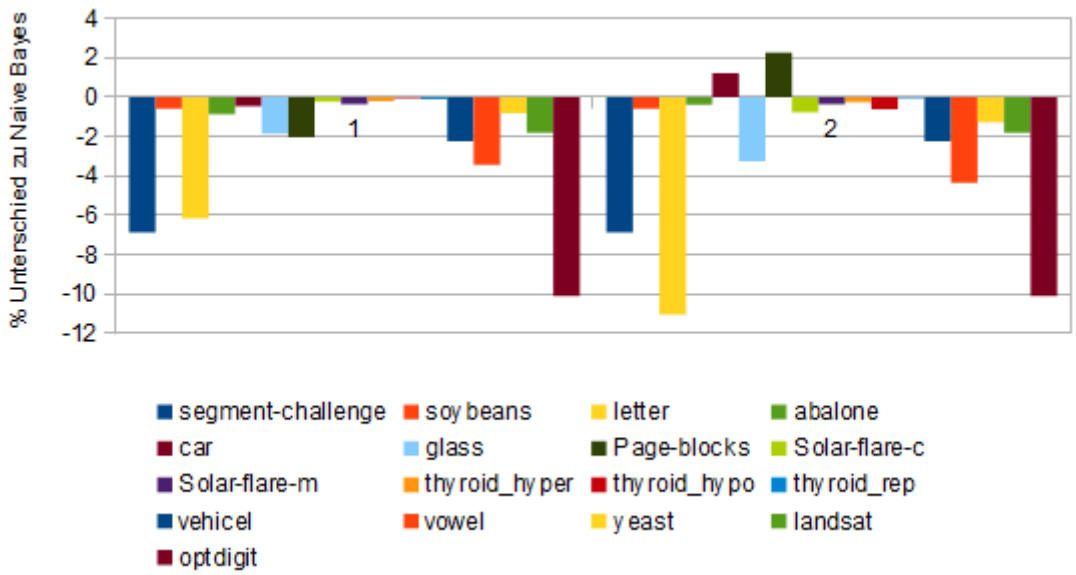


Abbildung 21: Genauigkeitsunterschiede von Naive Bayes zu CT -AB und -AB -R mit Naive Bayes

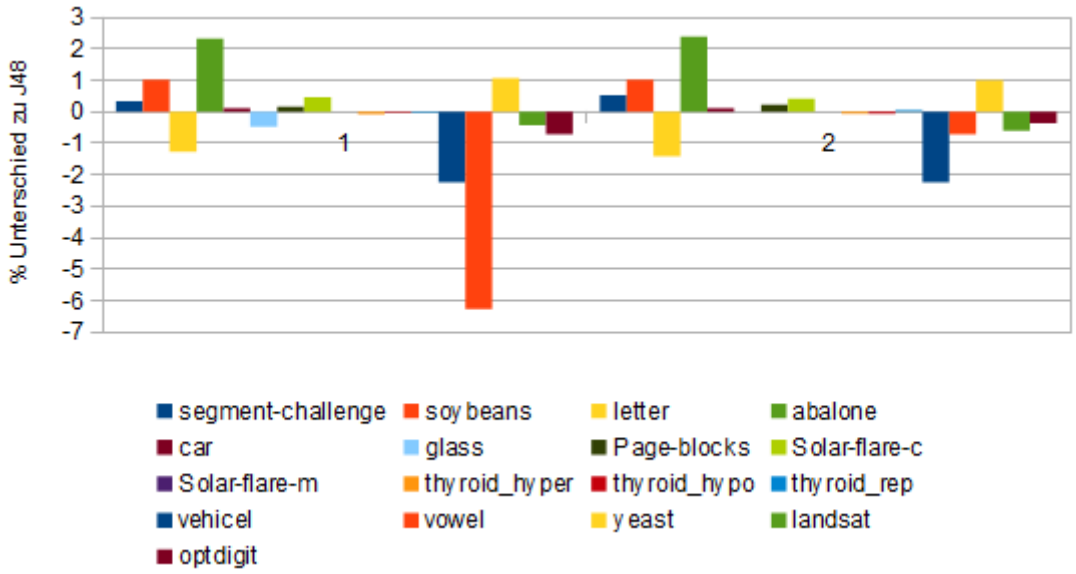


Abbildung 22: Genauigkeitsunterschiede von J48 zu CT -AB und -AB -R mit J48

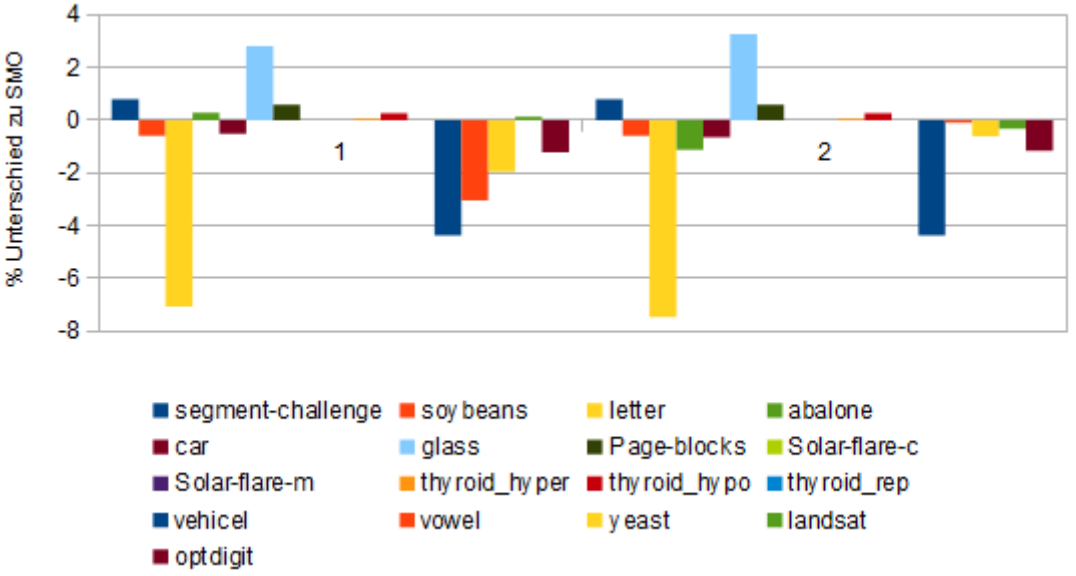


Abbildung 23: Genauigkeitsunterschiede von SMO zu CT -AB und -AB -R mit SMO

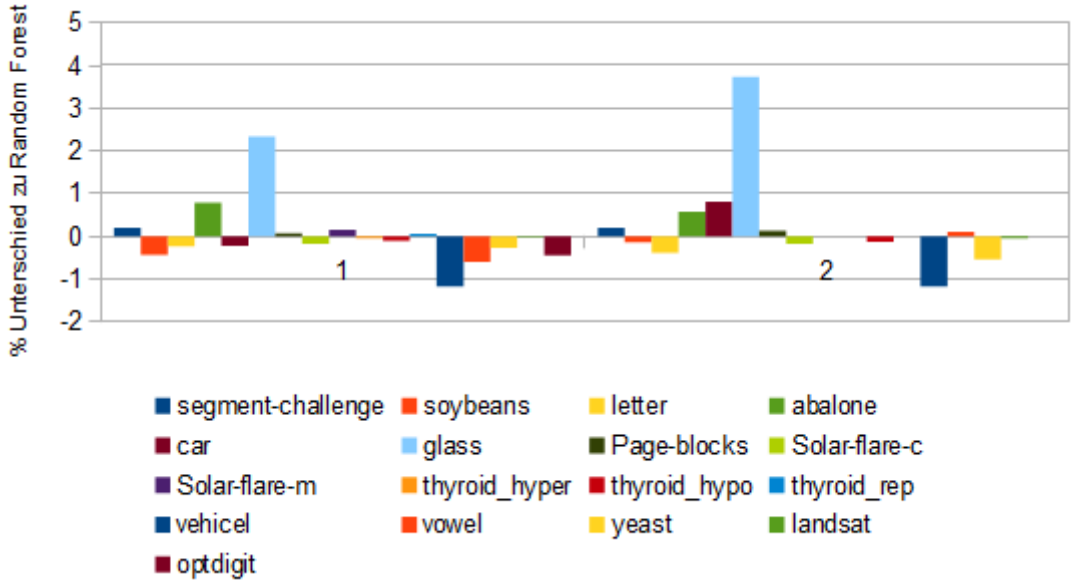


Abbildung 24: Genauigkeitsunterschiede von Random Forest zu CT -AB und -AB -R mit Random Forest

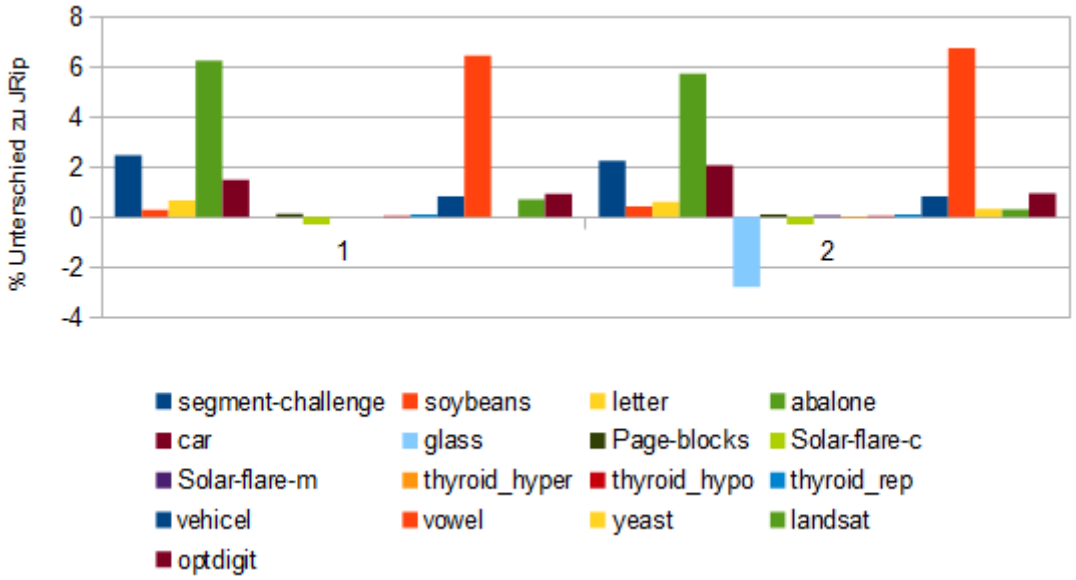


Abbildung 25: Genauigkeitsunterschiede von JRip zu CT -AB und -AB -R mit JRip

A.1.4 Balance mit Parameter

Tabelle 13: Genauigkeit der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu Naive Bayes auf den Testdatensätzen in Prozent

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	76,00	76,00	74,33	76,00	76,00	74,33
soybeans	81,84	82,58	80,97	84,48	83,75	80,82
letter	50,99	52,57	46,54	52,67	52,66	46,76
abalone	12,40	12,88	15,15	22,70	22,82	22,89
car	84,32	84,32	84,55	87,33	87,33	87,33
glass	38,79	38,79	39,72	39,25	39,72	45,79
Page-blocks	88,87	88,87	88,80	92,40	92,69	93,09
Solar-flare-c	80,20	80,20	79,67	77,22	77,45	79,09
Solar-flare-m	90,21	90,21	90,28	90,06	90,06	90,21
thyroid_hyper	95,41	95,41	95,28	94,99	95,016	95,12
thyroid_hypo	95,20	95,20	95,20	94,67	94,67	94,67
thyroid_rep	92,95	92,95	92,95	93,11	93,11	93,05
vehicel	41,13	42,55	42,55	42,55	42,55	42,55
vowel	57,88	59,70	60,10	59,90	59,60	59,29
yeast	57,08	56,60	56,13	56,40	56,94	55,86
landsat	77,07	77,32	76,01	77,32	77,32	76,0
optdigit	86,19	86,19	77,74	86,46	84,95	77,74

Tabelle 14: Genauigkeit der absoluten Variante und relativen mit Balance mit Parameter (-B x) von CT zu J48 auf den Testdatensätzen in Prozent

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	96,00	96,00	96,13	96,00	96,00	96,20
soybeans	92,09	92,39	92,68	92,24	92,39	92,68
letter	87,74	87,60	86,44	87,49	86,82	86,28
abalone	25,40	25,59	25,47	25,38	26,02	25,45
car	91,67	91,90	92,48	93,87	92,88	92,48
glass	70,09	70,09	68,69	67,29	65,89	66,36
Page-blocks	97,02	97,02	97,02	96,99	96,89	97,00
Solar-flare-c	85,57	85,57	85,57	85,51	85,51	85,51
Solar-flare-m	95,10	95,10	95,10	95,10	95,10	95,10
thyroid_hyper	98,83	98,83	98,83	98,86	98,86	98,86
thyroid_hypo	99,52	99,52	99,52	99,50	99,50	99,50
thyroid_rep	99,10	99,10	99,10	99,18	99,18	99,18
vehicel	70,09	70,09	70,21	70,09	70,33	70,21
vowel	79,80	79,29	74,24	80,40	79,39	78,38
yeast	58,83	58,83	58,42	58,56	59,03	57,68
landsat	85,19	85,41	85,98	85,70	85,50	85,86
optdigit	90,98	90,68	90,16	91,41	91,37	90,75

Tabelle 15: Genauigkeit der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu SMO auf den Testdatensätzen in Prozent

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	92,73	92,73	93,27	92,73	92,73	93,27
soybeans	93,70	93,70	93,56	93,56	93,56	93,56
letter	69,92	72,74	66,49	72,71	72,57	66,02
abalone	21,28	21,28	25,31	25,45	25,45	24,59
car	92,19	92,19	93,23	93,11	93,11	93,11
glass	56,07	56,07	58,88	58,41	59,35	59,35
Page-blocks	93,57	93,57	93,57	93,15	93,55	93,53
Solar-flare-c	85,16	85,16	85,16	85,16	85,16	85,16
Solar-flare-m	95,10	95,10	95,10	95,10	95,10	95,10
thyroid_hyper	97,88	97,88	97,88	97,91	97,91	97,91
thyroid_hypo	93,85	93,85	93,85	93,85	93,85	93,85
thyroid_rep	96,71	96,71	96,71	96,71	96,71	96,71
vehicel	74,11	74,11	69,98	74,11	74,11	69,98
vowel	74,14	78,18	59,80	78,18	76,87	63,94
yeast	51,68	51,68	54,72	55,59	55,66	56,33
landsat	87,058	87,058	87,13	87,19	87,19	86,25
optdigit	96,85	96,78	93,93	96,74	96,69	94,36

Tabelle 16: Genauigkeit der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu Random Forest auf den Testdatensätzen in Prozent

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	97,33	97,33	97,33	97,33	97,33	97,33
soybeans	90,92	90,92	90,92	91,07	91,07	91,07
letter	91,53	91,69	92,35	91,74	91,80	91,86
abalone	23,87	24,16	23,75	23,51	23,70	24,52
car	93,98	93,98	93,98	94,68	94,68	94,56
glass	76,17	76,17	76,17	77,10	77,10	76,17
Page-blocks	97,09	97,09	97,17	97,22	97,17	97,22
Solar-flare-c	84,17	84,17	84,11	84,35	84,35	84,11
Solar-flare-m	93,59	93,59	93,59	93,38	93,38	93,45
thyroid_hyper	98,70	98,70	98,70	98,73	98,73	98,73
thyroid_hypo	98,94	98,94	98,94	98,94	98,94	98,94
thyroid_rep	98,52	98,52	98,52	98,52	98,52	98,53
vehicel	75,30	75,30	75,30	75,30	75,30	75,30
vowel	94,24	94,24	94,24	94,55	94,55	94,55
yeast	57,55	57,41	58,56	57,82	58,09	57,82
landsat	89,36	89,45	89,76	89,45	89,49	89,99
optdigit	94,82	94,88	95,14	95,93	95,91	95,71

Tabelle 17: Genauigkeit der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu JRip auf den Testdatensätzen in Prozent

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	95,33	95,33	95,27	95,33	95,40	95,67
soybeans	93,41	93,41	92,68	93,12	92,97	93,12
letter	88,19	87,76	86,56	87,51	87,53	86,34
abalone	26,29	26,24	26,29	25,52	26,38	25,26
car	87,33	87,33	87,96	89,12	89,12	88,54
glass	69,16	69,16	63,08	66,82	65,89	64,49
Page-blocks	96,77	96,77	96,93	96,55	96,71	96,95
Solar-flare-c	85,11	85,11	85,11	85,11	85,11	85,11
Solar-flare-m	94,74	94,74	94,74	94,60	94,60	94,60
thyroid_hyper	98,54	98,54	98,57	98,49	98,49	98,49
thyroid_hypo	99,26	99,26	99,26	99,26	99,26	99,34
thyroid_rep	99,13	99,13	99,05	99,13	99,10	99,05
vehicel	69,39	69,39	70,57	69,39	69,39	70,57
vowel	76,57	76,46	76,36	76,46	76,77	76,36
yeast	57,21	57,08	57,35	57,61	57,41	57,75
landsat	86,74	86,83	86,31	86,65	86,76	86,18
optdigit	92,33	92,38	91,92	92,72	92,28	91,92

Tabelle 18: Differenz der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu Naive Bayes auf den Testdatensätzen

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	-5,07	-5,07	-6,73	-5,07	-5,07	-6,73
soybeans	-11,13	-10,40	-12,01	-8,49	-9,22	-12,15
letter	-13,13	-11,55	-17,58	-11,45	-11,46	-17,36
abalone	-11,44	-10,96	-8,69	-1,15	-1,03	-0,96
car	-1,22	-1,22	-0,98	1,79	1,79	1,79
glass	-9,81	-9,81	-8,88	-9,35	-8,88	-2,80
Page-blocks	-1,97	-1,97	-2,05	1,55	1,85	2,25
Solar-flare-c	0,29	0,29	-0,23	-2,69	-2,45	-0,82
Solar-flare-m	-0,58	-0,58	-0,50	-0,72	-0,72	-0,58
thyroid_hyper	0,027	0,027	-0,11	-0,40	-0,37	-0,27
thyroid_hypo	-0,05	-0,05	-0,05	-0,58	-0,58	-0,58
thyroid_rep	-0,16	-0,16	-0,16	0	0	-0,05
vehicel	-3,66	-2,25	-2,25	-2,25	-2,25	-2,25
vowel	-5,86	-4,04	-3,64	-3,84	-4,14	-4,44
yeast	-0,54	-1,01	-1,48	-1,21	-0,67	-1,75
landsat	-2,44	-2,19	-3,49	-2,19	-2,19	-3,49
optdigit	-5,14	-5,14	-13,59	-4,88	-6,39	-13,59

Tabelle 19: Differenz der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu J48 auf den Testdatensätzen

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	0,27	0,27	0,40	0,27	0,27	0,47
soybeans	0,59	0,88	1,17	0,73	0,88	1,17
letter	-0,24	-0,38	-1,55	-0,49	-1,17	-1,70
abalone	4,24	4,43	4,31	4,21	4,86	4,29
car	-0,69	-0,46	0,12	1,50	0,52	0,12
glass	3,27	3,27	1,87	0,47	-0,93	-0,47
Page-blocks	0,15	0,15	0,15	0,11	0,02	0,13
Solar-flare-c	0,47	0,47	0,47	0,41	0,41	0,41
Solar-flare-m	0	0	0	0	0	0
thyroid_hyper	-0,08	-0,08	-0,08	-0,05	-0,05	-0,05
thyroid_hypo	-0,027	-0,027	-0,027	-0,05	-0,05	-0,05
thyroid_rep	-0,027	-0,027	-0,027	0,05	0,05	0,05
vehicel	-2,36	-2,36	-2,25	-2,36	-2,13	-2,25
vowel	-1,72	-2,22	-7,27	-1,11	-2,12	-3,13
yeast	2,83	2,83	2,43	2,56	3,03	1,68
landsat	-0,97	-0,74	-0,18	-0,45	-0,65	-0,29
optdigit	0,28	-0,02	-0,53	0,71	0,68	0,05

Tabelle 20: Differenz der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu SMO auf den Testdatensätzen

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	0,80	0,80	1,33	0,80	0,80	1,33
soybeans	-0,15	-0,15	-0,29	-0,29	-0,29	-0,29
letter	-12,43	-9,60	-15,86	-9,64	-9,78	-16,32
abalone	-3,97	-3,97	0,05	0,19	0,19	-0,67
car	-1,56	-1,56	-0,52	-0,64	-0,64	-0,64
glass	0	0	2,80	2,34	3,27	3,27
Page-blocks	0,64	0,64	0,64	0,22	0,62	0,60
Solar-flare-c	0	0	0	0	0	0
Solar-flare-m	0	0	0	0	0	0
thyroid_hyper	0,05	0,05	0,05	0,08	0,08	0,08
thyroid_hypo	0,27	0,27	0,27	0,27	0,27	0,27
thyroid_rep	-0,03	-0,03	-0,03	-0,03	-0,03	-0,03
vehicel	-0,24	-0,24	-4,37	-0,24	-0,24	-4,37
vowel	2,73	6,77	-11,62	6,77	5,45	-7,47
yeast	-5,39	-5,39	-2,36	-1,48	-1,42	-0,74
landsat	0,50	0,50	0,56	0,63	0,63	-0,32
optdigit	-1,48	-1,55	-4,40	-1,58	-1,64	-3,97

Tabelle 21: Differenz der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu Random Forest auf den Testdatensätzen

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	0,67	0,67	0,67	0,67	0,67	0,67
soybeans	-0,29	-0,29	-0,29	-0,15	-0,15	-0,15
letter	-2,45	-2,29	-1,62	-2,23	-2,18	-2,12
abalone	0,79	1,08	0,67	0,43	0,62	1,44
car	0	0	0	0,69	0,69	0,58
glass	1,87	1,87	1,87	2,80	2,80	1,87
Page-blocks	-0,11	-0,11	-0,04	0,02	-0,04	0,02
Solar-flare-c	-0,23	-0,23	-0,29	-0,06	-0,06	-0,29
Solar-flare-m	0,14	0,14	0,14	-0,07	-0,07	0
thyroid_hyper	0,03	0,03	0,03	0,05	0,05	0,05
thyroid_hypo	-0,13	-0,13	-0,13	-0,13	-0,13	-0,13
thyroid_rep	-0,05	-0,05	-0,05	-0,05	-0,05	-0,05
vehicel	-0,71	-0,71	-0,71	-0,71	-0,71	-0,71
vowel	-0,40	-0,40	-0,40	-0,10	-0,10	-0,10
yeast	-1,21	-1,35	-0,20	-0,94	-0,67	-0,94
landsat	-0,56	-0,47	-0,16	-0,47	-0,43	0,07
optdigit	-1,92	-1,87	-1,60	-0,82	-0,84	-1,03

Tabelle 22: Differenz der absoluten und relativen Variante mit Balance mit Parameter (-B x) von CT zu JRip auf den Testdatensätzen

Datensatz	-B 0,1	-B 1,0	-B 10,0	-R -B 0,1	-R -B 1,0	-R -B 10,0
segment-challenge	2,20	2,20	2,13	2,20	2,27	2,53
soybeans	1,17	1,17	0,44	0,88	0,73	0,88
letter	2,16	1,73	0,52	1,47	1,50	0,30
abalone	6,92	6,87	6,92	6,15	7,01	5,89
car	0,87	0,87	1,50	2,66	2,66	2,08
glass	0,47	0,47	-5,61	-1,87	-2,80	-4,21
Page-blocks	-0,07	-0,07	0,09	-0,29	-0,13	0,11
Solar-flare-c	-0,29	-0,29	-0,29	-0,29	-0,29	-0,29
Solar-flare-m	0	0	0	-0,14	-0,14	-0,14
thyroid_hyper	0,05	0,05	0,08	0	0	0
thyroid_hypo	-0,08	-0,08	-0,08	-0,08	-0,08	0
thyroid_rep	0,19	0,19	0,11	0,19	0,16	0,11
vehicel	0,35	0,35	1,54	0,35	0,35	1,54
vowel	6,46	6,36	6,26	6,36	6,67	6,26
yeast	-0,88	-1,01	-0,74	-0,47	-0,67	-0,34
landsat	1,10	1,20	0,68	1,01	1,13	0,54
optdigit	1,62	1,67	1,21	2,01	1,57	1,21

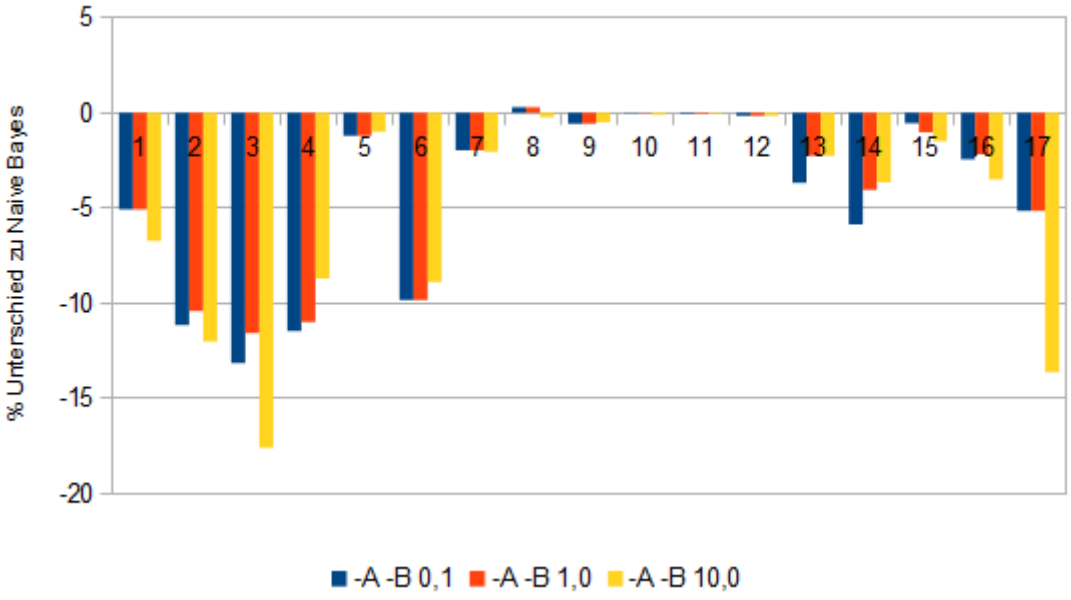


Abbildung 26: Genauigkeitsunterschiede von Naive Bayes zu CT -B 0.1 , 1.0 und 10.0 mit Naive Bayes

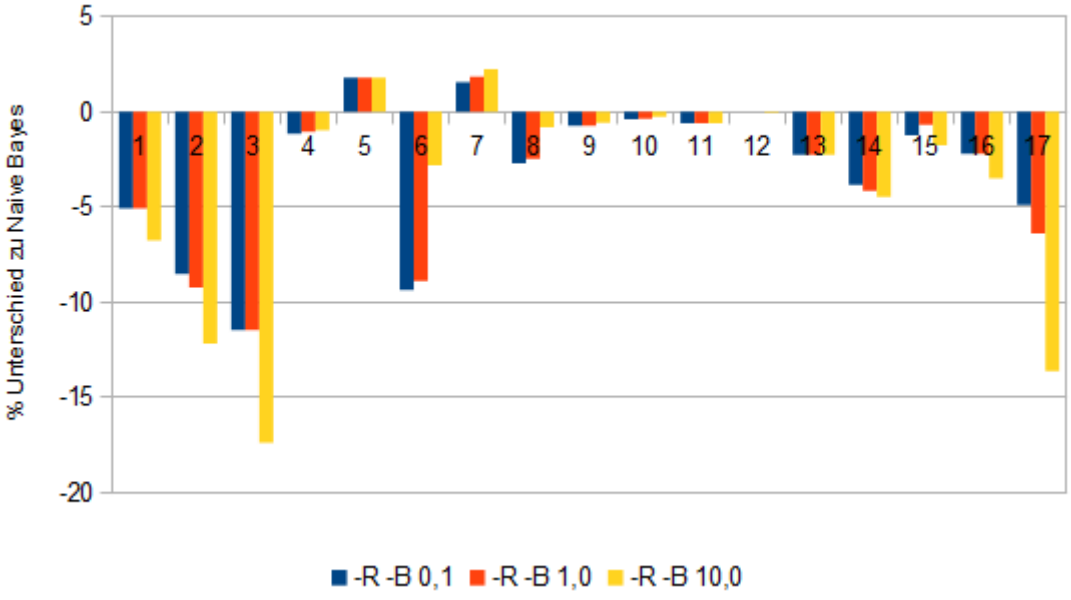


Abbildung 27: Genauigkeitsunterschiede von Naive Bayes zu CT -R -B 0.1 , 1.0 und 10.0 mit Naive Bayes

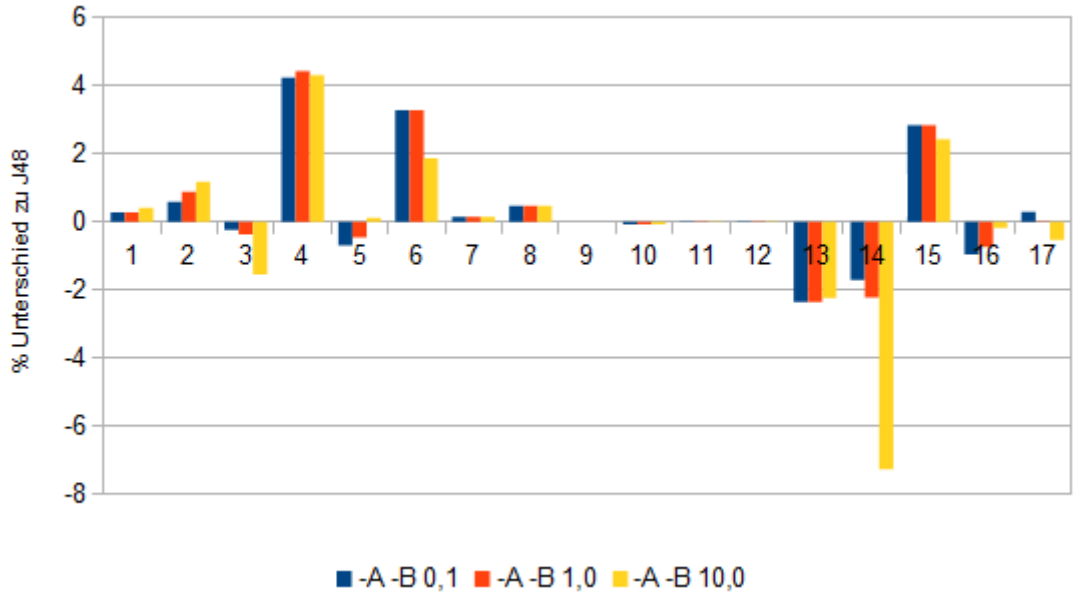


Abbildung 28: Genauigkeitsunterschiede von J48 zu CT -B 0.1 , 1.0 und 10.0 mit J48

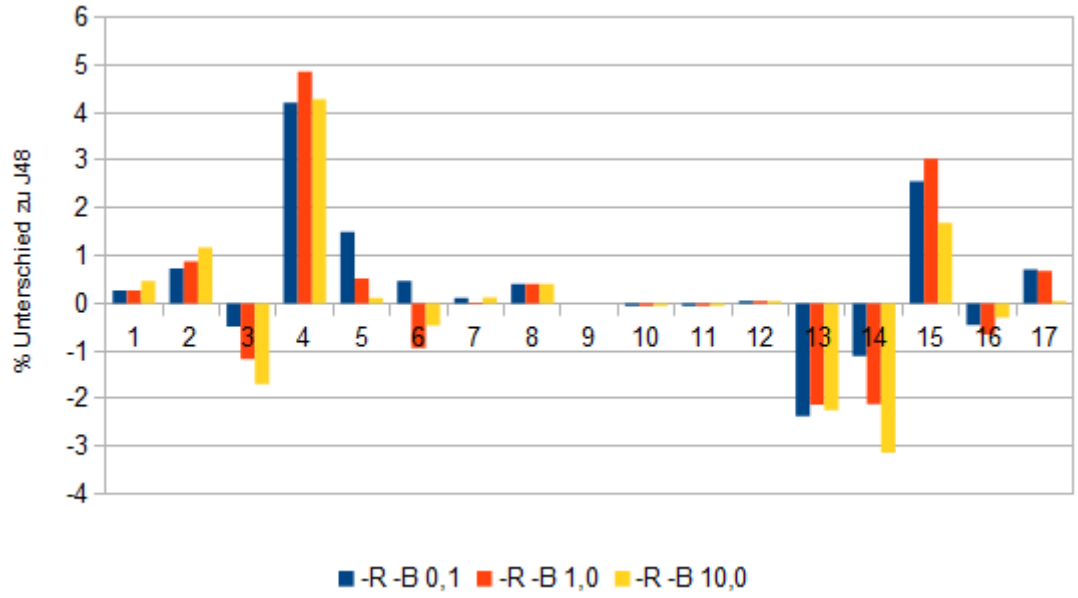


Abbildung 29: Genauigkeitsunterschiede von J48 zu CT -R -B 0.1 , 1.0 und 10.0 mit J48

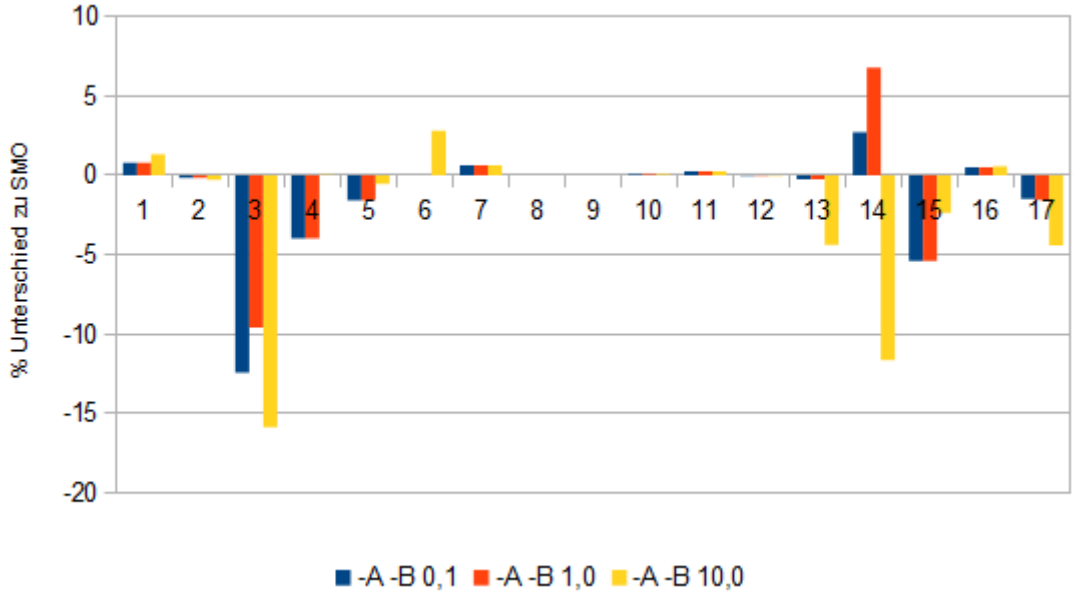


Abbildung 30: Genauigkeitsunterschiede von SMO zu CT -B 0.1 , 1.0 und 10.0 mit SMO

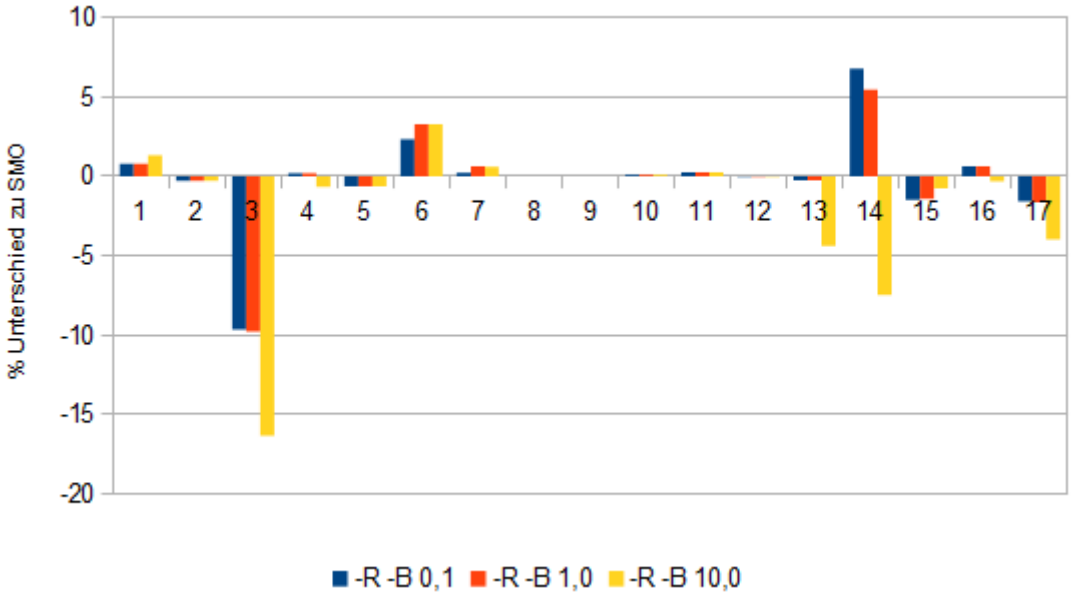


Abbildung 31: Genauigkeitsunterschiede von SMO zu CT -R -B 0.1 , 1.0 und 10.0 mit SMO

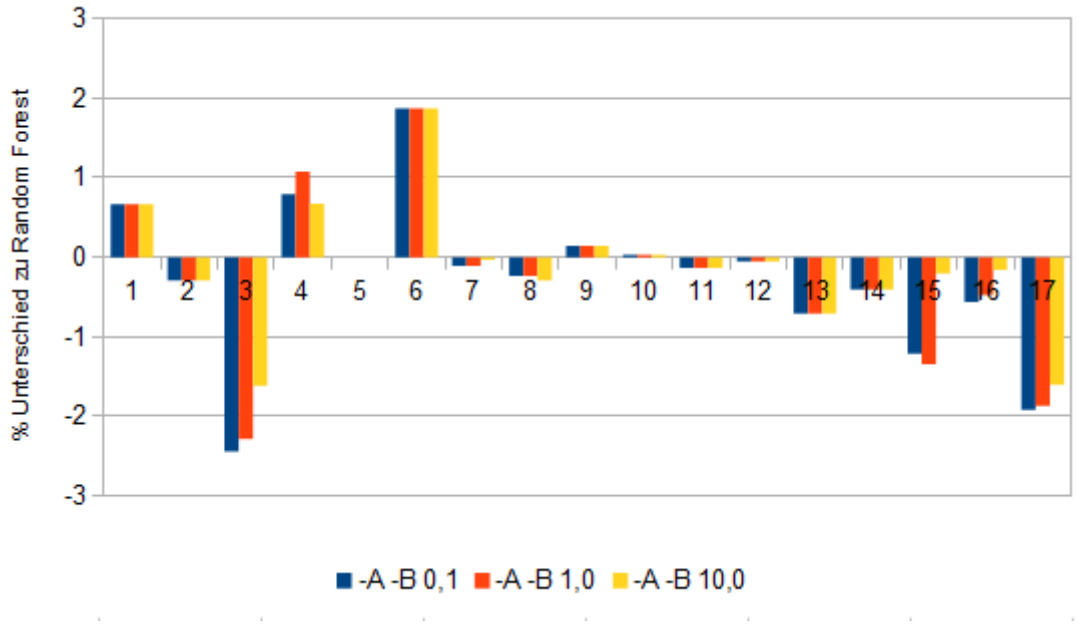


Abbildung 32: Genauigkeitsunterschiede von Random Forest zu CT -B 0.1 , 1.0 und 10.0 mit Random Forest

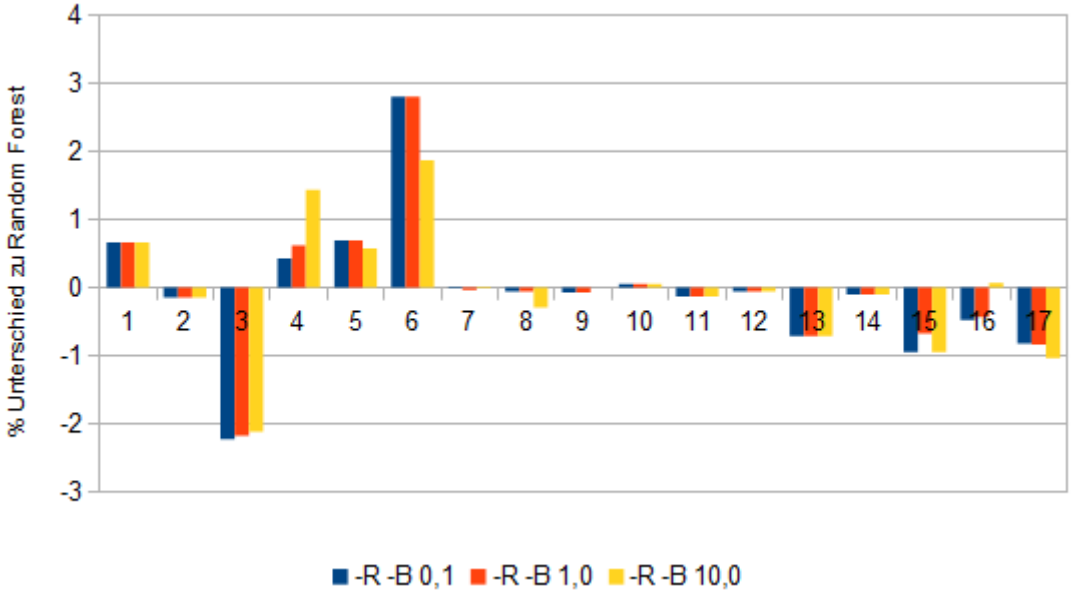


Abbildung 33: Genauigkeitsunterschiede von Random Forest zu CT -R -B 0.1 , 1.0 und 10.0 mit Random Forest

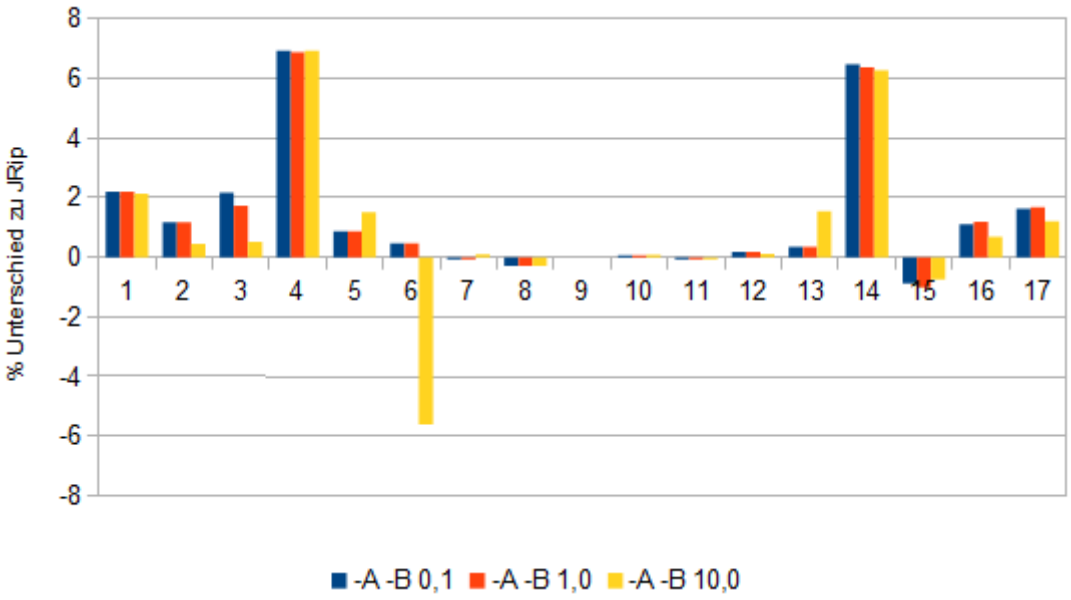


Abbildung 34: Genauigkeitsunterschiede von JRip zu CT -B 0.1 , 1.0 und 10.0 mit JRip

A.2 Abbildungen der Baumstrukturen

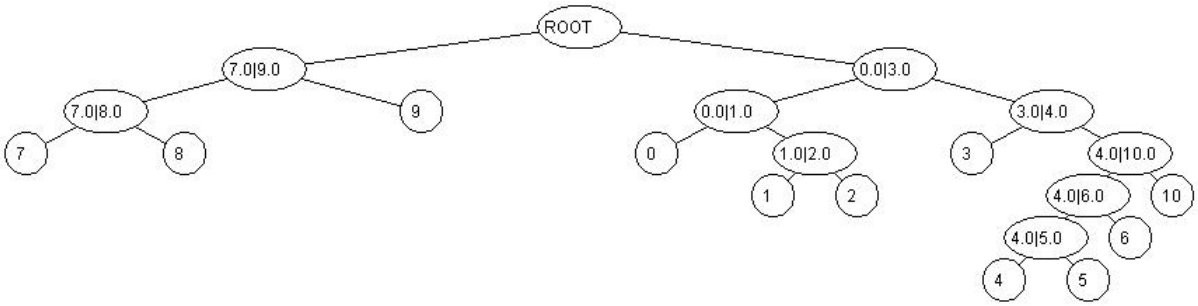


Abbildung 35: Baum des Datensatzes vowel von CT -R mit SMO

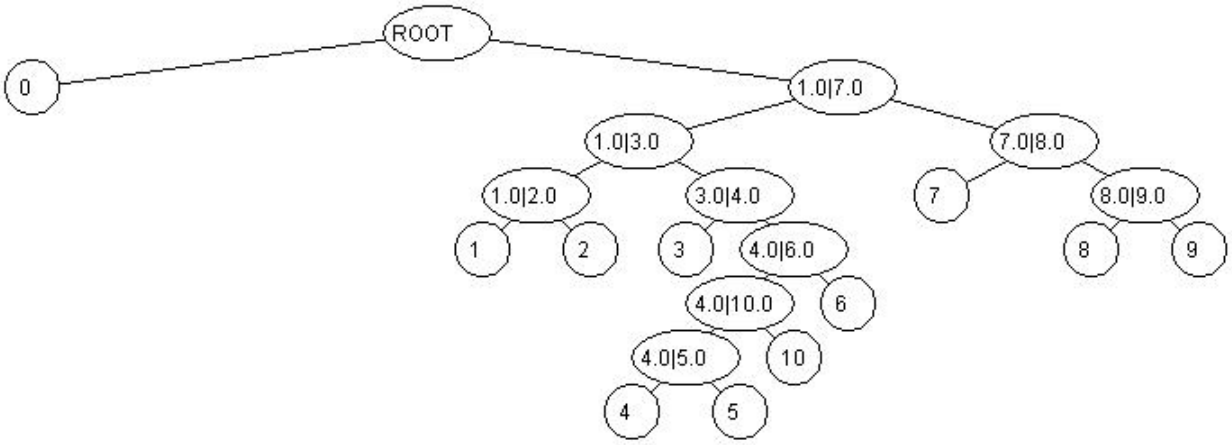


Abbildung 36: Baum des Datensatzes vowel von CT -R mit J48

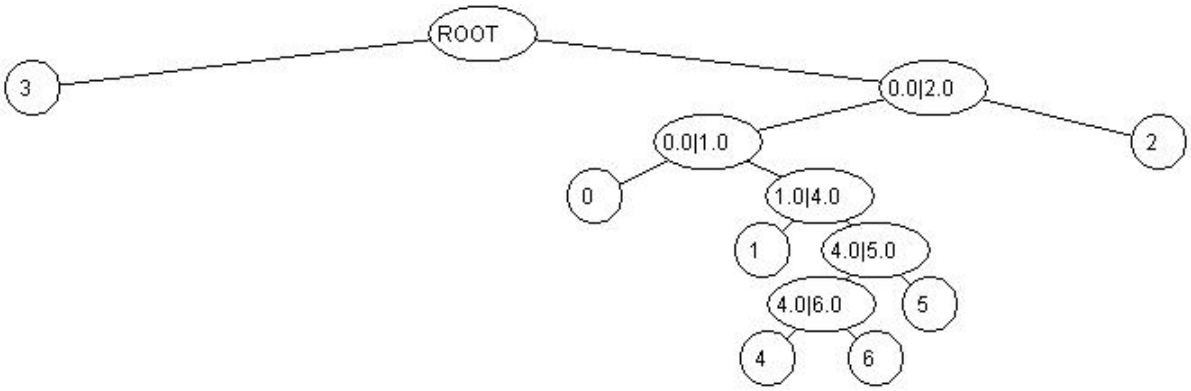


Abbildung 37: Baum des Datensatzes glass von CT -R mit J48

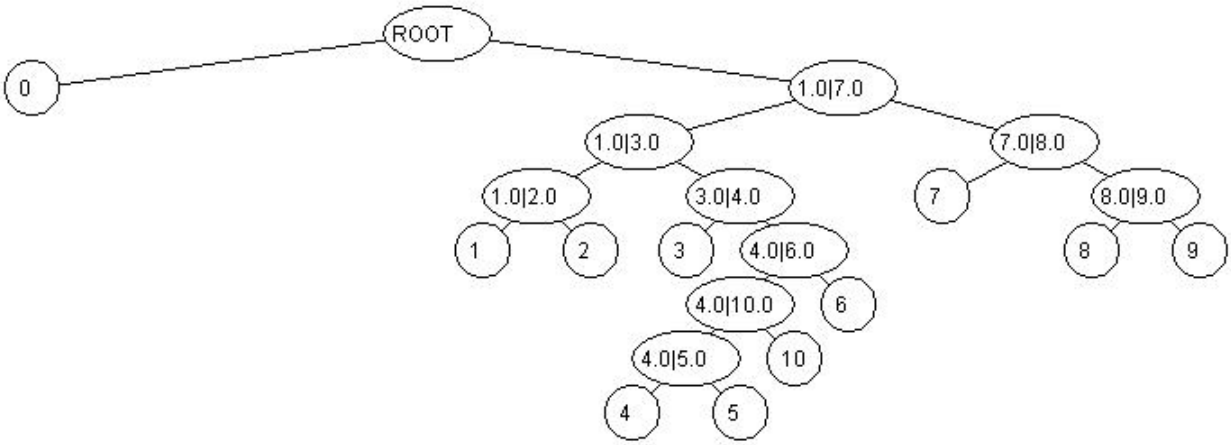


Abbildung 38: Baum des Datensatzes vowel von CT -R -B 0,1 mit J48

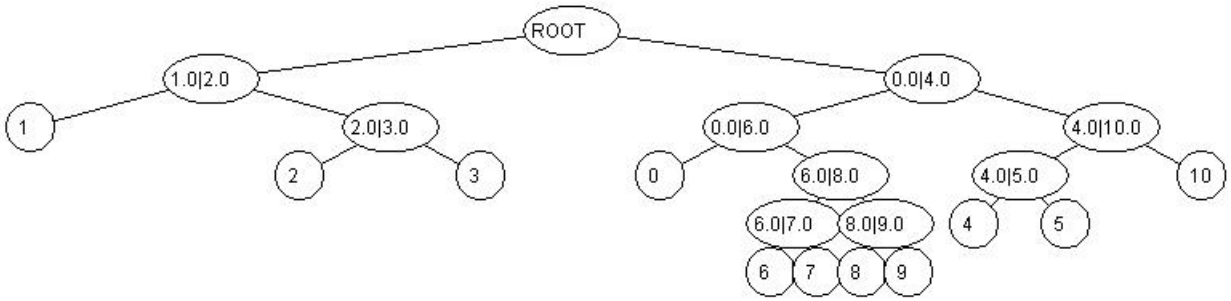


Abbildung 39: Baum des Datensatzes vowel von CT -R -B 1,0 mit J48

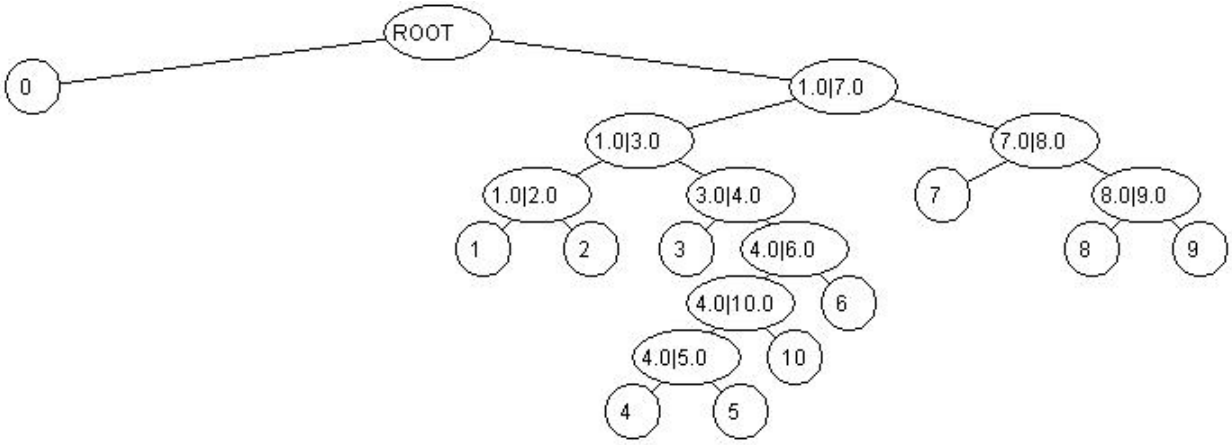


Abbildung 40: Baum des Datensatzes vowel von CT -R -B 10,0 mit J48

B Abkürzungsverzeichnis

CT	Classifier Trees
DAGSVM	Directed Acyclic Graph Support Vektor Maschine
DB2	divide-by-2
DDAG	Decision Directed Acyclic Graph
ID3	Iterative Dichotomiser 3
IREP	incremental reduced error pruning
NB	Naive Bayes
Ripper	Repeated Incremental Pruning to Produce Error Reduction
RF	Random Forest
ROC	Receiver Operating Characteristic
SMO	Sequential Minimal Optimization
SVM	Support Vektor Maschine
weka	Waikato Environment for Knowledge Analysis

Literatur

[1] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 163–171. Curran Associates, Inc., 2010.

[2] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] Rich Caruana and Alexandru Niculescu-mizil. An empirical comparison of supervised learning algorithms. In *Proc. 23 rd Intl. Conf. Machine learning (ICML'06*, pages 161–168, 2006.

[4] William W. Cohen. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[5] Pat Langley George H. John. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo*, pages 338–345, 1995.

[6] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. SIGKDD Explorations, Volume 11, Issue 1, 2009.

[7] Baoyuan Liu, F. Sadeghi, M. Tappen, O. Shamir, and Ce Liu. Probabilistic label trees for efficient large scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 843–850, June 2013.

[8] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[9] John C. Platt, Nello Cristianini, and John Shawe-taylor. Large margin dags for multiclass classification. In *Advances in Neural Information Processing Systems 12*, pages 547–553, 2000.

[10] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[11] Volkan Vural and Jennifer G. Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 105–112, New York, NY, USA, 2004. ACM.