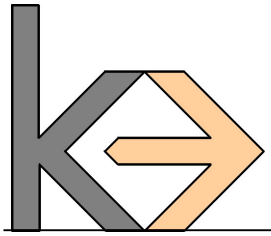


TECHNISCHE UNIVERSITÄT DARMSTADT

FACHBEREICH INFORMATIK
Knowledge Engineering

Diplomarbeit

Effiziente Klassifikation und Ranking mit paarweisen Vergleichen

Sang-Hyeun Park
13. Dezember 2006

Betreuer
Prof. Dr. J. Fürnkranz

Technische Universität Darmstadt
Fachbereich Informatik
Hochschulstrasse 10
64289 Darmstadt

Ehrenwörtliche Erklärung

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

Sang-Hyeun Park
Darmstadt, 13. Dezember 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzzusammenfassung

Die Round-Robin Binärisierung hat neben vielen Vorteilen den großen Nachteil, dass die Klassifikationsphase quadratisch in der Anzahl der Klassen ist. Im Hinblick auf Multiklassifikation und Rankingbestimmung wurde untersucht, inwieweit man die Effizienz steigern könnte. Ansatzpunkt für effiziente Rankings war speziell das Schweizer-System aus dem Schach. Die Ergebnisse für das Schweizer-System wiesen auf ein lineares Verhältnis zwischen Genauigkeitsverlust und Vergleichsreduzierung auf. Jedoch ist die Reihenfolgeart des Schweizer-Systems einer zufälligen oder (klassen-)geordneten Reihenfolgeart überlegen. Daneben wird aber für die Klassifikation ein effizienter Dekodierungsalgorithmus (*Quick Weighted*) vorgestellt, dass auf Weighted Voting basiert. Mittels empirischer Auswertungen wird gezeigt, dass Quick Weighted eine durchschnittliche Laufzeit von $n \cdot \log(n)$ besitzt, wobei das Klassifikationsergebnis jedoch identisch mit Weighted Voting (quadratische Laufzeit) ist.

1 Einleitung

1.1 Einleitung und Motivation

Das heutige Zeitalter wird oft als Informationszeitalter bezeichnet. Mit dem Einzug der Computer in nahezu allen Bereichen der heutigen Gesellschaft, entstand eine unübersichtliche Fülle an digitalen Informationen. Der Erfolg von Suchmaschinen war somit vorprogrammiert. Nicht nur für das World Wide Web bedarf es einer Suchmaschine, der eigene Computer stellt eine eigene Welt dar.

Der Begriff Suchmaschine erweckt dabei eine falsche Vorstellung von der Arbeitsebene. Informationen werden syntaktisch oder auf statistischer Ebene ausgewertet. Es wird nach Kriterien wie übereinstimmende Zeichenketten oder Anzahl der Vorkommen eines Wortes das World Wide Web durchsucht. Welche Bedeutung diese Informationen jedoch haben, also die semantische Ebene der Informationen, ist der Suchmaschine nicht erschließbar.

Jedoch wurden im Laufe der Geschichte der interdisziplinären Forschungsrichtung „Künstliche Intelligenz“ Werkzeuge entwickelt, die auf die Vision hin arbeiten, vorsichtig ausgedrückt, den Computer etwas intelligenter zu machen.

In letzter Zeit konnte ein Boom des Maschinellen Lernens beobachtet werden, der sich unter dem Begriff „Data Mining“ manifestierte. Anhand der heutigen großen verfügbaren Menge an digitalen Informationen werden diese dabei mittels rechnerbasierter Lernverfahren nach bisher unbekannten Zusammenhängen sowie Beziehungen zwischen Objekten und Gesetzmässigkeiten durchforstet. Diese Entwicklung ist wohl besonders auf ihre vielfältigen Anwendungsgebiete, insbesondere in der Wirtschaft, zurückzuführen.

Eines der grundlegenden Werkzeuge des Maschinellen Lernens stellt dabei die maschinelle Klassifikation dar. Mit maschinellen Lernverfahren, die auf Trainingsdaten angewendet werden, ist es möglich sogenannte Klassifikatoren zu erzeugen, die eine Einteilung von Testobjekten in Klassen vornehmen. Mittels dieser Klassifikatoren ist es auch möglich, eine nach Wahrscheinlichkeit bzw. Relevanz geordnete Liste der Klassen - ein Ranking - zu bestimmen. An dem Klassifikationsproblem wird schon seit Längerem geforscht und verständlicherweise zählten die zwei Eigenschaften Genauigkeit und Effizienz immer zu den wesentlichen Gesichtspunkten.

Bei Multiklassifikationsproblemen, also Problemen mit mehr als zwei möglichen Klassen, wird üblicherweise eine sogenannte Binärisierung der Daten vollzogen. Die Daten werden in eine Menge von Teildaten mit jeweils zwei Klassen aufgespaltet und anhand dieser Daten paarweise Klassifikator gelernt. Die paarweise Binärisierung erzeugt dabei relativ gute Ergebnisse bezüglich der Genauigkeit gegenüber den anderen Binärisierungsarten. Diese Verbesserung der Genauigkeit schlägt sich aber negativ auf die Effizienz nieder.

Die vorliegende Arbeit versuchte nach Möglichkeiten zu suchen, die diesen Nachteil der paarweisen Binärisierung reduzieren könnten. Dabei dienten unter anderem verschiedene Reihenfolgen der Klassifikatoren und kombinatorische Aspekte einer Stimmenverteilung als Hilfsmittel.

1.2 Gliederung

In Kapitel 2 wird ein kurzer Überblick über die Thematik wiedergegeben und wichtige Begriffe definiert.

Kapitel 3 beschäftigt sich mit der effizienten Klassifikation. Dabei werden zunächst gängige Dekodierungsmethoden vorgestellt und danach auf den Algorithmus *Quick Weighted Voting* eingegangen. Nach der Erläuterung der Hauptidee des Algorithmus anhand eines Beispiels werden einige Eigenschaften beschrieben und die empirischen Auswertungen präsentiert. Außerdem wird in einem eigenen Unterkapitel kurz auf die Frage eingegangen, in wiefern der Algorithmus als Suchalgorithmus einzuordnen ist.

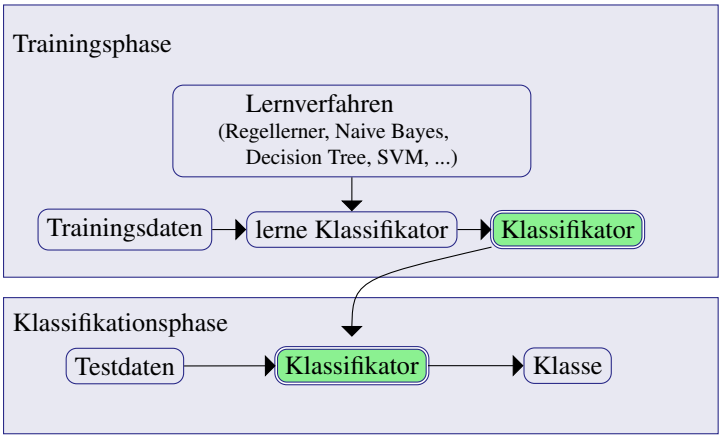
Effizientes Ranking mit paarweisen Vergleichen wird in Kapitel 4 beschrieben. Hier wird auf das Schweizer-System eingegangen und deren Auswertungsergebnisse präsentiert. Das Schweizer-System wird um sogenannte Feinwertungen (Sonneborn-Berger, Buchholz) erweitert und wieder an verschiedenen Datensätzen ausgewertet. Abschließend werden weitergehende Statistiken präsentiert, die zur Beurteilung der Feinwertungen dienen.

In Kapitel 5 folgt dann eine Zusammenfassung und kurzes Fazit der Diplomarbeit.

2 Grundlagen

2.1 maschinelle Klassifikation

Die *maschinelle* Klassifikation ist ein Teilgebiet aus dem Maschinellen Lernen. Dabei geht es um den Vorgang der *automatischen* Klassifizierung von Objekten mittels gelernter Verfahren. Im Allgemeinen wird eine Funktion, der *Klassifikator*, mithilfe eines Lernverfahrens zunächst aus Beispielobjekten, den sogenannten *Trainingsdaten* „gelernt“. Als Ergebnis dieses Lernvorgangs steht ein *Klassifikator* zur Verfügung, der auf neue Objekte, die *Testdaten,* angewendet werden kann.



Die Klassifikation ist dabei in zwei Phasen - Trainingsphase und Klassifikationsphase - aufgeteilt. In obiger Abbildung ist eine schematische Darstellung dazu zu sehen.

Definition 2.1 (Klassifikation, Klassifikator) *Sei* $K = \{k_1, k_2, k_3, \ldots, k_n\}$ *die Menge der Klassen,* $T = \{t_1, t_2, t_3, \ldots, t_m\}$ *die Menge der Daten. Gesucht ist eine Funktion (***Klassifikator***)* $f : T \rightarrow K$*, die jedem Testobjekt eine Klasse zuordnet und bestimmte Kriterien erfüllt.*

Die in Definition 2.1 nicht näher definierten Kriterien sollen informal ausgedrückt die „wahrscheinlichste“ oder „beste“ Klasse garantieren. Konkrete Kriterien ergeben sich erst im Hinblick auf das verwendete Lernverfahren.

2.2 Lernverfahren

Es gibt mehrere Verfahren, die auf unterschiedliche Arten „lernen“. Im Folgenden soll ein kurzer Überblick über Lernverfahren gegeben werden, die unter anderem für die Klassifikation benutzt werden. Als Grundlage dienten hauptsächlich die Bücher von Witten [25] und Mitchell [16] und es wurden jeweils nur die Grundkonzepte dargestellt. Für weitergehende Informationen bieten sich die aufgeführten Literaturverweise an.

2.2.1 Entscheidungsbäume

Entscheidungsbäume sind im Kontext der Klassifikation Bäume, die an den inneren Knoten Tests beinhalten und deren Blätter Klassen darstellen. Für die Klassifikation einer Instanz wird beginnend von der Wurzel ein Pfad im Baum verfolgt, der durch die Tests vorgegeben ist. Das erreichte Blatt entspricht schließlich dem Klassifikationsergebnis. Ein einfaches Beispiel ist in Abbildung 2.1 zu sehen.

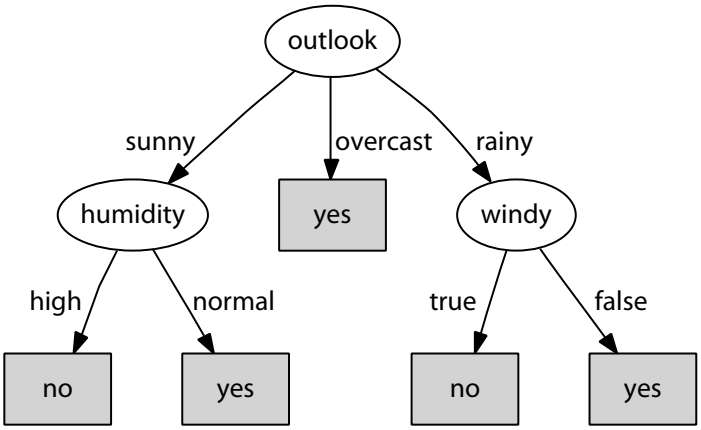


Abbildung 2.1: Einfacher Entscheidungsbaum anhand des `weather` Datensatzes. Der Entscheidungsbaum soll mithilfe der Wettervorhersage dazu dienen eine Entscheidung zu treffen, ob ein Spiel z. B. Golf gespielt werden sollte (Quelle: [25]).

Entscheidungsbaumlernen basiert auf dem „divide and conquer“ Ansatz. Die gegebenen Trainingsdaten werden anhand eines Attributs geteilt. Diese Teilung wird durch einen Test als Knoten im Baum festgehalten. Je nach Anzahl der möglichen Testergebnisse (Ja/Nein, (1,2,3,4)) ergibt sich daraufhin eine Menge an gerichteten Kanten. Angenommen der Test t_1 teile die Trainingsdaten in I und J auf. Der Knoten t_1 besitzt dann zwei ausgehende Kanten zu t_2 und t_3. Sollten alle Trainingsdaten aus I einer Klasse k zugeordnet sein, so ist der Knoten t_2 ein Blatt mit dem Wert k, analog für J. Falls nicht, wird der Teilungsvorgang auf I und J wiederholt. Dies stellt das Grundprinzip des Entscheidungsbaumlernens dar.

Es existieren viele Stellen, die Fragestellungen aufwerfen. Nach welchem Attribut sollte aufgeteilt werden oder gibt es gar einen Maßstab für ein gutes Teilungsattribut? Werden größere Bäume bevorzugt, oder doch Kleinere? Es werden dafür Eigenschaften wie der Information-Gain, der eine Aussage über die erwartete Verringerung der Entropie einer Aufteilung eines bestimmten Attributs liefert, oder den sogenannten Gini-Index [1] herangezogen.

Für das Problem des sogenannten „Overfitting“, die Unflexibilität der Entscheidungsbäume auf neue Instanzen aufgrund überangepasster Klassifikatoren, existieren Pre- und Post-Pruning Methoden.

Die prominentesten Vertreter der Entscheidungsbaumlerner stellen unter anderem CARTs [1] und ID3/C4.5 [19] dar.

2.2.2 Regellernen

Regellerner bilden die Klassifikationsfunktion als eine Menge von Regeln ab.

IF Bedingung THEN Klasse a

Die Generierung von Regelmengen kann einerseits durch die Abbildung eines Entscheidungsbaumes auf Regeln vollzogen werden oder andererseits durch Verfahren, die direkt Regelmengen bilden. Ein naives Verfahren wäre es, jede einzelne Instanz der Trainingsdaten auf eine Regel abzubilden. Für das Lernen von Regeln wird üblicherweise nach dem „separate and conquer“ Prinzip (siehe Algorithmus 1) vorgegangen.

Algorithmus 1 : Separate and Conquer

Data : leere Regelmenge T und Trainingsdaten E

1 **begin**
2 Lerne eine einzelne Regel R aus E und füge sie zu T hinzu.
3 **if** T *ist akzeptabel* **then**
4 return T
5 **else**
6 SEPARATE: Entferne Instanzen aus E, die von R abgedeckt werden.
7 CONQUER: WENN E nicht leer ist, gehe zu 2.
8 **end**

Einzelne Regeln können beginnend mit einer allgemeinen Regel, die alle Instanzen abdeckt, schrittweise spezialisiert werden, indem Bedingungen hinzugefügt werden. Neben diesem top-down Ansatz existiert die umgekehrte Version, die von einer speziellen Regel beginnend, mehr und mehr verallgemeinert wird. Auch die Kombination beider Arten ist als bidirektionales Verfahren möglich.

Die Regellernmethoden sind einer der ältesten maschinellen Lernverfahren. Im Laufe der Zeit wurde eine Vielzahl von Verfahren entwickelt, die auf diesem Prinzip basieren. Dabei lassen sie sich durch folgende Eigenschaften unterscheiden:

- **Hypothesensprache**
  Die verwendete Hypothesensprache der Regeln definiert die Repräsentationsfähigkeit. Zum Beispiel können durch Verwendung von logischen Aussagen erster Ordnung eleganter Regeln definiert werden als mit der Propositionslogik.

- **Regel Such-Algorithmus**
  Für die Bildung einer Regel können verschiedene Algorithmen wie z. B. Hill-Climbing, Best-First oder ein stochastisches Verfahren angewendet werden. Daneben ist die „Richtung“ (top-down, bottom-up, bidirektional) der Algorithmen unterschiedlich.

- **Behandlung von Overfitting**
  Einige Regellernverfahren bieten keine Gegenmaßnahmen zum Overfitting-Problem an, wobei bei anderen mittels Pre- oder Post-Pruning dagegen vorgehen. Es existieren sogar einige Verfahren mit bereits integrierten Vorbeugemaßnahmen.

2.2.3 Support-Vector-Maschinen

Die Klassifikation mit *Support-Vector-Maschinen* (SVM) beruht auf der Idee, eine Hyperebene zu bestimmen, die die Klassen unter verschiedenen Kriterien optimal trennt.

Sind die Klassen von einem Trainingsdatensatz linear separierbar, dann existieren in der Regel eine unendliche Menge an möglichen Hyperebenen. Support-Vector-Maschinen unterscheiden sich von anderen Lernverfahren nun dadurch, dass diejenige Hyperebene ausgewählt wird, sodass der kleinste Abstand einer Trainingsinstanz zur Hyperebene maximal wird. Bei Benutzung der euklidischen Norm entsteht dabei ein quadratisches Optimierungsproblem.

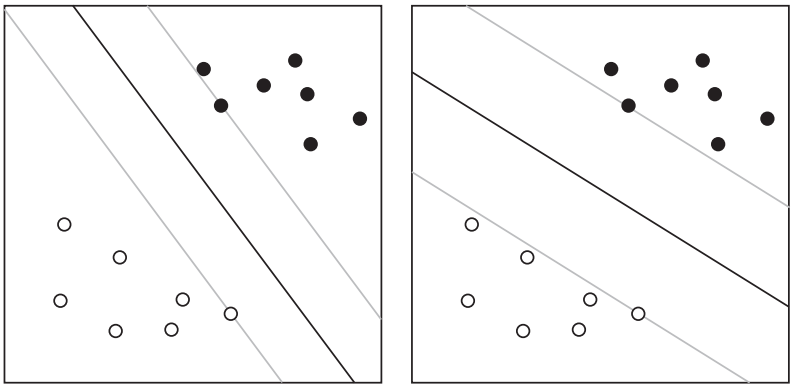


Abbildung 2.2: SVM: Hyperebene diskriminiert zwei Klassen (Quelle: [2])

Sollte der Datensatz nicht linear separierbar sein, so werden die Daten in einen Raum höherer Dimension abgebildet. Aufgrund des Theorems von Cover [21] wird durch eine derartige Abbildung die Anzahl möglich linearer Trennungen erhöht. Mithilfe sogenannter Kernelfunktionen [21] lassen sich Hyperebenen in einem hochdimensionalen Raum implizit berechnen, sodass die Kosten gering sind.

2.2.4 Bayes Learning

Bei Bayes Learning handelt es sich um den wahrscheinlichkeitstheoretischen Ansatz der Klassifikation. Das heißt mit Mitteln der Wahrscheinlichkeitstheorie wird die Hypothese bzw. Klasse $k_j \in K$ bestimmt, die die größte Wahrscheinlichkeit unter den Testdaten hat. Dabei spielt der *Satz von Bayes* eine zentrale Rolle, da er das Rechnen mit bedingten Wahrscheinlichkeiten ermöglicht. Der Satz von Bayes lautet folgendermaßen:

$$P(h|D) = \frac{P(D|h) * P(h)}{P(D)}$$

wobei h für Hypothese bzw. in diesem Fall Klasse und D für Daten steht

Die Wahrscheinlichkeiten $P(h)$ nennt man die „a priori“ Wahrscheinlichkeit von h und $P(h|D)$ die „a posteriori“. Der Grund dafür ist, dass Erstere die Wahrscheinlichkeit von h unabhängig von anderen Ereignissen beschreibt, und Letztere nachdem die Testdaten beobachtet wurden.

Bei der Klassifikation ist man bekanntlich interessiert an der unter den Testdaten wahrscheinlichsten Klasse. Diese Klasse nennt man *„maximum a posteriori“* (MAP) Klasse. Mithilfe des Bayes-Theorem lassen sich für jede Klasse

die Wahrscheinlichkeiten ausrechnen und somit die Wahrscheinlichste bestimmen.

$$k_{MAP} = arg\max_{k \in K} P(k|D) \tag{2.1}$$

$$= arg\max_{k \in K} \frac{P(D|k) \cdot P(k)}{P(D)} \tag{2.2}$$

$$= arg\max_{k \in K} P(D|k) \cdot P(k) \tag{2.3}$$

In letzten Schritt wurde die Gleichung vereinfacht, indem der Nenner entfernt wurde, da er unabhängig von k ist.

Auf dieser Grundlage existiert eine Vielzahl von Lernalgorithmen. Genauer gesagt spricht man von einem *Bayes optimalen* Klassifizierer, wenn der Lernprozess nach obigen Gleichungen verfährt. Da es jedoch sehr aufwendig sein kann, für jede neue zu klassifizierende Instanz die „a posteriori“ Wahrscheinlichkeiten zu bestimmen, wurden Varianten entwickelt.

Der *Naive Bayes* Klassifizierer ist zwar eine weniger optimale Variante, jedoch ist sie sehr effizient. Sie basiert auf der Annahme, dass die bedingten Wahrscheinlichkeiten der Attributwerte von Daten $D = (a_1, \cdots, a_n)$ unabhängig sind unter einer gegebenen Klasse k. Das heißt: $P(a_1, \cdots, a_n|k) = \prod_i P(a_i|k)$. Damit wird aus Gleichung (2.3) die vereinfachte Gleichung :

$$k_{NB} = arg\max_{k \in K} P(k) \prod_i P(a_i|k)$$

Im Allgemeinen ist die Unabhängigkeitsannahme jedoch nicht erfüllt. Die *AODE* [24] (averaged, one-dependence estimators) geht von einer schwächeren Unabhängigkeitsannahme als der Naive Bayes aus. Dabei werden mehrere verschiedene Teilmengen der Attribute als unabhängig betrachtet und damit Klassifikationsauswertungen berechnet. Aus diesen mehreren Modellen wird daraufhin das Endergebnis durch Durchschnittsbildung ermittelt.

Die Verfahren, in denen die Unabhängigkeitsannahme nur auf Teilmengen der Attributwerte angenommen wird, werden unter dem Begriff Bayes Networks oder Bayesian Belief Networks zusammengefasst [7].

2.3 Multiklassifikation

Man spricht von Multiklassifikation, wenn die betrachteten Datensätze in mehr als zwei Klassen kategorisiert werden können. Diese Problemstellung wird gesondert betrachtet, da viele Lernverfahren auf 2-Klassenprobleme beschränkt sind. In [8] wurde dieser Sachverhalt erläutert. Darin wurden mehrere Gründe aufgezählt:

- Beschränkung durch die Hypothesensprache
  Beispiele dafür sind Lineare Diskriminanten oder Support-Vector-Maschinen.
- Beschränkung durch die Lernarchitektur
  Z. B. Neuronale Netze mit einem singulären Output
- Beschränkung durch das Lern-Framework

Für die Behandlung solcher Mehrklassenprobleme gibt es zwei Möglichkeiten. Entweder werden die Lernalgorithmen verallgemeinert, sodass *direkt* Multiklassenprobleme behandelt werden können oder die binären Lernverfahren

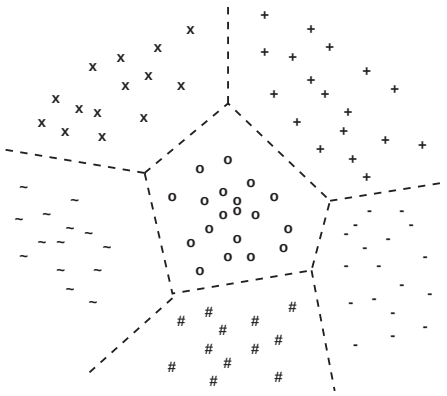


Abbildung 2.3: Multiklassifikationsproblem: Das Problem besteht aus 6 Klassen, die mit verschiedenen Symbolen dargestellt sind (Quelle: [8]).

bleiben ungeändert und es findet eine *Binärisierung* der Daten vor dem Lernvorgang statt.

2.3.1 Binärisierungsarten

Dieser Überblick über die Binärisierungsarten wurde hauptsächlich aus [8] übernommen. Alle Binärisierungsarten haben die Eigenschaft ein Multiklassifikationsproblem in eine Reihe von *binären* Klassifikationsproblemen umzuwandeln. Dies geschieht durch eine Art Filterung der Daten, bevor sie den binären Lernalgorithmen als Eingabe dienen.

Ungeordnete Binärisierung

Definition 2.2 *Die ungeordnete Binärisierung wandelt ein n-Klassenproblem zu n binären Klassenproblemen um. Dabei werden die Trainingsinstanzen der Klasse i als positiv und alle Trainingsinstanzen der Klasse j als negativ markiert ($j = 1 \ldots n,\ j \neq i$).*

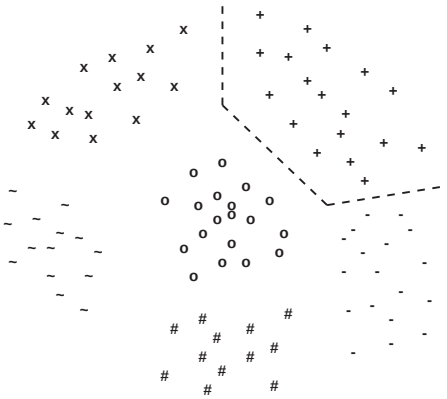


Abbildung 2.4: Ungeordnete Binärisierung Das Problem besteht aus 6 Klassen, es entstehen 6 Klassifikatoren, jeder diskriminiert eine Klasse (in diesem Fall Klasse „+“) von allen anderen Klassen (Quelle: [8]).

Die ungeordnete Binärisierung (siehe Abbildung 2.4) ist in der Literatur unter anderem auch als *One-versus-All* (OVA) oder *One-versus-Rest* (OVR) Methode bekannt.

Geordnete Binärisierung

Definition 2.3 *Die geordnete Binärisierung wandelt ein n-Klassenproblem zu $n-1$ binären Klassenproblemen um. Dabei werden die Trainingsinstanzen der Klasse i ($i = 1 \ldots c-1$) als positiv und alle Trainingsinstanzen der Klasse $j > i$ als negativ markiert.*

Die Idee oder der Vorteil der geordneten Binärisierung wird einem klar, wenn man sich das Dekodierungsverfahren mit den ungeordneten und geordneten Verfahren bewusst macht. Nach der Trainingsphase mit der ungeordneten Binärisierung hat man Klassifikatoren zur Verfügung, welche auf eine Testinstanz angewendet aussagen, ob sie entweder zu Klasse i gehören oder nicht. Das heißt, man geht in keiner bestimmten Reihenfolge die Klassifikatoren durch, bis ein Klassifikator ein positives Ergebnis bestätigt, im schlimmsten Falle[1] also alle n Klassifikatoren. Es wäre wünschenswert eine Heuristik für die Reihenfolge zu haben, die diese Anzahl an Klassifikatorauswertungen reduziert. Genau diese Überlegung findet man bei der geordneten Binärisierung. Dazu benötigt man eine gewisse Ordnung der Klassen. Typischerweise ist die Ordnung durch die Instanzenanzahl bestimmt.

Definition 2.4 (Ordnung von Klassen) *Sei #i die Anzahl der Trainingsinstanzen, die als Klasse i markiert ist. Dann gilt für die Ordnung $O()$ der Klassen $j,k = 1 \ldots n,\ j \neq k$:*

$$O(j) < O(k) \Leftrightarrow \#j < \#k$$

Das heißt man interpretiert die Trainingsinstanzen als eine gute Stichprobe, in dem Sinne, dass das Häufigkeitsverhältnis der Trainingsinstanzen von bestimmten Klassen, also die Ordnung, sich auch in den Testdaten widerspiegeln wird.

Mit dieser Ordnung hat man nun eine gewisse Heuristik, mit der man die durchschnittliche Anzahl der Klassifikatorauswertungen reduzieren kann. Man beginnt bei dem Klassifikator mit der „größten“ Ordnungszahl, also die Klasse, die am häufigsten in den Trainingsdaten vorkam, und wertet die nächstkleinere Klasse aus, falls das Ergebnis nicht positiv war. Wenn man davon ausgeht, dass sich das Häufigkeitsverhältnis der Klassen von Trainingsdaten und Testdaten ähnelt, dann ist ersichtlich, dass man in der Regel beim geordneten Verfahren weniger Auswertungen benötigt.

Diese Ordnung bietet nicht nur in der Klassifikationsphase einen Vorteil, sondern auch in der Trainingsphase. Anders als beim ungeordneten Verfahren werden nicht alle Trainingsinstanzen für das Lernen eines Klassifikators hinzugezogen (außer bei der größten Klasse). Aufgrund der Ordnung ist die Reihenfolge der Klassifikatorauswertungen in der Dekodierung von vornherein festgelegt (beginnend von der größten bis zur kleinsten). Wenn die Auswertung für die größte Klasse negativ war, dann wird die nächstkleinere ausgewertet. Bei der nächstkleineren Klassenauswertung ist aber wiederum klar, dass die Testinstanz nicht der größten Klasse zugehörig sein kann, da dies davor schon geprüft wurde. Das heißt, man kann für das Lernen von Klassifikatoren für Klassen einer bestimmten Position in der Ordnung die Testdaten mit den größeren Klassen auslassen. Somit werden gegenüber der ungeordneten Binärisierung weniger Daten beim Lernverfahren ausgewertet und dies führt insgesamt zu einer Reduzierung der Laufzeit.

Diese Art der Binärisierung findet unter anderem Verwendung bei dem bekannten Regellerner RIPPER [3].

[1] Es sind im Worst-Case n Klassifikatoren auszuwerten, unter der Annahme, dass Testinstanzen existieren, die keiner Klasse zugeordnet werden können. Unter der Einschränkung, dass solche Fälle nicht vorkommen, sind maximal $n-1$ Klassifikatoren auszuwerten.

Round-Robin Binärisierung

Definition 2.5 (Single RR) *Die Round-Robin Binärisierung (auch paarweise Vergleiche genannt) wandelt ein n-Klassenproblem zu $\frac{n(n-1)}{2}$ binären Klassenproblemen $< i, j >$ um. Jeweils eines für ein Paar der Klassen $i, j \in \{1, \ldots, n\}$ mit $i < j$. Der binäre Klassifikator für das Problem $< i, j >$ wird mithilfe der Trainingsinstanzen der Klassen i und j gelernt. Alle restlichen Trainingsdaten werden für dieses Problem ignoriert.*

Streng genommen beschreibt obige Definition das *Single Round-Robin* Verfahren. Daneben existiert das *Double Round-Robin* Verfahren. Lockert man in obiger Definition die Bedingung, dass für jeweils ein binäres Klassenproblem $< i, j > \Leftrightarrow i < j$ zu $< i, j > \Leftrightarrow i \neq j$ gelten muss, ergeben sich doppelt so viele binäre Klassenprobleme und entsprechende Klassifikatoren. Dies ist die sogenannte *Double Round-Robin* Binärisierung. Der Sinn liegt darin, dass einige Lernverfahren nicht *Klassen-symmetrisch* sind. Das heißt, $< i, j >$ und $< j, i >$ sind zwei verschiedene Klassifikationsprobleme. Viele Lernverfahren wählen nämlich eine Klasse als Standard Klasse aus und lernen nur Regeln, um die andere Klasse abzudecken.

Definition 2.6 (Double RR) *Die Double Round-Robin Binärisierung wandelt ein n-Klassenproblem zu $n(n-1)$ binären Klassenproblemen $< i, j >$ um. Jeweils eines für ein Paar der Klassen $i, j \in \{1, \ldots, n\}$ mit $i \neq j$. Der binäre Klassifikator für das Problem $< i, j >$ wird mithilfe der Trainingsinstanzen der Klassen i und j gelernt. Alle restlichen Trainingsdaten werden für dieses Problem ignoriert.*

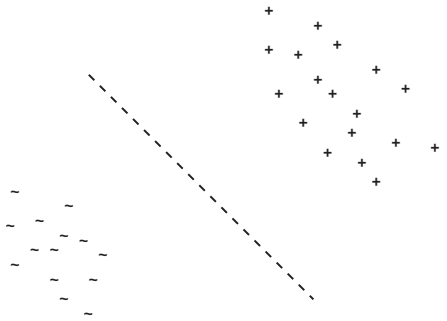


Abbildung 2.5: Round-Robin Binärisierung $\frac{n(n-1)}{2}$ Klassifizierer. Jeweils eines für ein Paar von Klassen (Quelle: [8]).

Da die Round-Robin Binärisierung für diese Diplomarbeit einen Schwerpunkt bildet, wird im Folgenden näher auf seine Eigenschaften eingegangen.

Die Round-Robin Binärisierung liefert eine Menge an Vorteilen gegenüber den anderen Binärisierungsarten. Folgende Auflistung der Eigenschaften basiert wieder auf den Untersuchungen von Fürnkranz in [8].

- Genauigkeit
  - In der Auswertung von 20 Datensätzen hat RR nie gegen *1-vs-rest* verloren.
  - Die Ergebnisse waren sogar oft signifikant besser (basierend auf McNemar Test).

- Effizienz
  - Es ist beweisbar schneller als *1-vs-rest*, ECOC, boosting,...
  - Die Effizienzsteigerung ist umso größer für langsamere Basislerner[2].
- Einfachheit/Verständnis
  - Es werden einfachere Klassifikatoren generiert, die im Hinblick für das Konzeptlernen von Vorteil sind.
- Datenreduktion
  - Es wird eine wesentlich geringere Trainingsmenge für jedes Binärproblem benutzt als für das ursprüngliche Problem.
  - Dies kann den Vorteil haben, dass Teilprobleme komplett in den Speicher passen, wohingegen es beim ursprünglichen Problem nicht möglich war.
- Parallelisierbarkeit
  - Da jeder einzelne binäre Lernvorgang unabhängig voneinander ist, eignet sich Round-Robin für parallele Trainingsphasen.

Round-Robin Binärisierung hat also gegenüber den anderen Binärisierungsarten einige Vorteile, jedoch hat es seinen größten Nachteil in der Klassifikationsphase. Aufgrund der quadratischen Anzahl der erzeugten binären Probleme und somit Klassifikatoren ergeben sich für Dekodierungsalgorithmen, die alle Klassifikatoren auswerten, eine quadratische Laufzeit. Die ungeordnete und geordnete Binärisierung benötigt jedoch nur eine lineare Laufzeit.

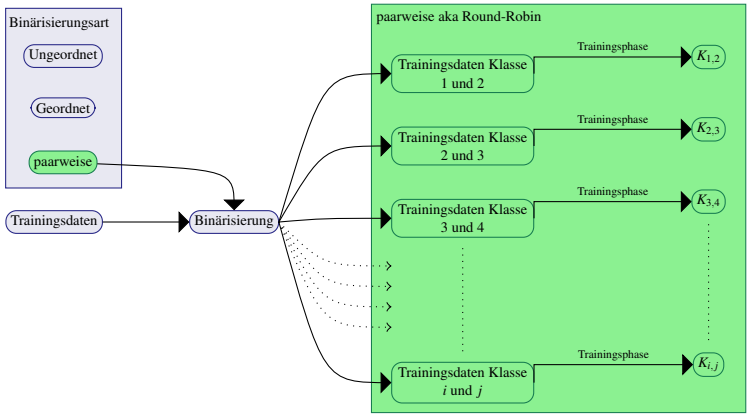


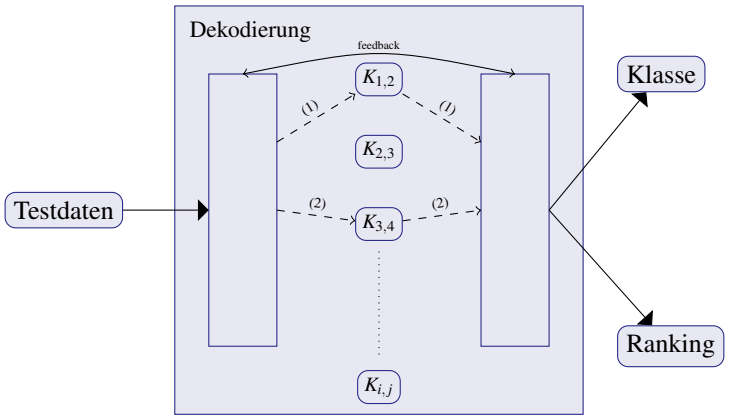
Abbildung 2.6: Round-Robin Binärisierung: Schematische Darstellung der Trainingsphase

Es wird darauf hingewiesen, dass in den folgenden Kapiteln bei Verwendung des Begriffs „Klassifikator“ ein paarweiser Klassifikator assoziiert werden soll. Alle folgenden Aussagen und Auswertungen bezüglich Klassifikation und Ranking basieren in dieser Diplomarbeit auf paarweise Vergleiche bzw. Round-Robin Binärisierung.

[2] Unter Basislerner versteht man das Lernverfahren, das für die binären Klassifikationsprobleme benutzt wurde.

2.3.2 Dekodierung

Durch die Binärisierung ist es nun zwar möglich bisherige (binäre) Lernverfahren auf Multiklassifikationsprobleme anzuwenden, jedoch steht man vor dem neuen Problem auf welche Art man die Menge der Klassifikatoren kombinieren sollte, um eine Vorhersage zu treffen. Dieser neue Abschnitt, der bei der Multiklassifikation auftritt, bezeichnet man als Dekodierungsphase. In folgender Abbildung ist sie schematisch dargestellt.



Die Dekodierung ist dabei in der Mitte als Blackbox dargestellt. Nach Eingabe der Testdaten kann vor einer Klassifikatorauswertung schon eine Art Preprocessing stattfinden, wie z. B. eine Sortierung der Klassifikatoren nach Ordnung der Klassen[3]. Die gestrichelten Pfeile mit Zahlenangaben sollen die Reihenfolge der Klassifikatorauswertungen andeuten. Somit wird in der obigen Abbildung zunächst der Klassifikator $K_{1,2}$ angewendet bzw. ausgewertet. Daraufhin könnte ein Dekodierungsalgorithmus je nach Klassifikatorergebnis von $K_{1,2}$ Statistiken, die für den Auswahlprozess des nächsten Klassifikators relevant sind[4], aktualisieren. Als Ergebnis sei die Auswahl auf den Klassifikator $K_{3,4}$ gefallen.

Das Schema erscheint für die meisten Dekodierungsalgorithmen komplexer als notwendig. Später wird auch deutlich, dass einer der prominentesten Dekodierungsarten *max wins* oder *Voting* vollkommen unabhängig von dem vorherigen Klassifikationsergebnis bei der Abarbeitung der Klassifikatoren verfährt.

Da der Fokus dieser Diplomarbeit jedoch auf eine effiziente Dekodierungsphase hin gerichtet ist, wurde die Dekodierungsphase strukturierter dargestellt, um bei der Entwicklung einer effizienteren Dekodierung hilfreich zu sein.

Bisher wurde die Reihenfolge einer Dekodierung geschildert. Welche verschiedenen Verfahren letztendlich für die Bestimmung einer einheitlichen Vorhersage existieren, wird in 3.1 und 4.2 vorgestellt.

Analogie zu Spielen oder Turniersystemen

Der Dekodierungsprozess stellt eine große Ähnlichkeit zu Turnieren im Sport oder in Spielen dar. Bei der Klassifikation wird nach der Klasse gesucht, die je nach Kriterium oder Lernverfahren am besten abschneidet. Bei Spielen trifft

[3] Dies wird bei der geordneten Binärisierungsart mit der *max-wins* Dekodierung benötigt.

[4] Es werden später Dekodierungsalgorithmen vorgestellt, die durch Heuristiken, basierend auf vorhergehenden Klassifikatorauswertungen, Klassifikatoren „überspringen“. Dabei spielen die Informationen wie z. B. Anzahl der bisherigen Klassifikatorauswertungen für jede Klassen oder vorläufige Punkteverlauf eine Rolle

dies im übertragenen Sinne auch zu. Aus diesem Grund werden häufig Beispiele aus Spielen herangezogen oder Begriffe aus diesem Bereich synonym verwendet. Der paarweise Klassifikator $K_{1,2}$ mit dem Ergebnis, dass Klasse 1 vorzuziehen ist, wird zu: $Spieler_1$ gewinnt die Partie $Spieler_1$ vs. $Spieler_2$.

2.4 Ranking

Beim Ranking-Problem geht man von der selben Ausgangstellung wie bei der Klassifikation aus. Jedoch wird nicht allein die „beste“ oder „wahrscheinlichste“ Klasse gesucht, sondern eine vollständige Liste der Klassen, absteigend geordnet nach der Relevanz. Ging es bei der Klassifikation noch um die Abbildung der Testdaten auf eine Klasse, so stellt die Rankinggenerierung eine Abbildung auf eine geordnete Menge von Klassen dar. Insofern könnte man das Ranking-Problem als Verallgemeinerung der Klassifikation ansehen.

Definition 2.7 (Ranking-Problem) *Sei $K = \{k_1, k_2, k_3, \ldots, k_n\}$ die Menge der Klassen, $T = \{t_1, t_2, t_3, \ldots, t_m\}$ die Menge der Daten. Gesucht ist eine Funktion $f : T \rightarrow K^n$, die jedem Testobjekt eine absteigend nach Relevanz geordnete Liste von Klassen zuordnet.*

Die Rankinggenerierung findet in der Regel mithilfe von gelernten Klassifikatoren statt. Die Klasse an erster Stelle des Rankings sollte idealerweise das Ergebnis einer Klassifikation sein, sofern identische Lernverfahren oder Dekodierungsmethoden verwendet werden. Bei unterschiedlichen Lernverfahren ist es möglich unterschiedliche Ergebnisse zu erhalten. Diese „beste“ oder „wahrscheinlichste“ Klasse wird auch als *top rank* bezeichnet. Dabei wird dieser Begriff auch im Zusammenhang der Klassifikation benutzt. Das heißt, die Klasse, die das Klassifikationsergebnis auf bestimmten Daten darstellt, wird auch als „top rank“ bezeichnet.

Definition 2.8 (top rank) *Sei R ein Ranking von Klassen. Die Klasse an Position eins des Rankings wird als* **top rank** *bezeichnet.*

3 Effiziente Klassifikation

Ansatzpunkt für eine effiziente Klassifikation war in dieser Diplomarbeit die Dekodierungsphase. Das heißt, es wurde untersucht, ob eine effiziente Methode auf der Metaebene existiert, sodass eine Reduktion der Vergleichszahl möglich ist. Mit der Metaebene ist dabei gemeint, dass die Methode unabhängig vom verwendeten Basislerner ist.

3.1 Überblick der Dekodierungsarten

3.1.1 (Weighted) Voting

Voting, oft auch als *max wins* bezeichnet, ist eine sehr simple Dekodierungsmethode. Es basiert auf einem einfachen Prinzip:

- Jeder binäre Klassifikator stimmt für eine Klasse.
- Die Klasse mit den meisten Stimmen ist der *top rank*

Definition 3.1 (Voting, Voting-for) *Sei $K_{i,j} : T \rightarrow [0,1]$ der binäre Klassifikator für die Klassen k_i, k_j und t die zu klassifizierende Testinstanz.*

$$top\ rank := arg \max_{i \in K} \sum_{j \in K} K_{i,j}(t)$$

Für den Fall, dass der *top rank* nicht eindeutig ist, gibt es mehrere Lösungen diesen Gleichstandszustand zu umgehen. Eine Möglichkeit ist es, die Klasse mit der größten Ordnungszahl (Definition 2.4) auszuwählen.

Definition 3.2 (Gleichstandsbehandlung 1) *Sei P die Menge der Klassen, die maximale Punktzahlen haben und O die Ordnungszahl aus 2.4. Dann gilt*

$$top\ rank := arg \max_{i \in P} O(i)$$

Da die Ordnungszahl wiederum auch nicht immer eindeutig ist, wird üblicherweise als letzte Instanz eine zufällige Wahl getroffen.

Definition 3.3 (Gleichstandsbehandlung 2) *Sei Q die Menge der Klassen, die aus 3.2 entstanden ist. Dann wähle zufällig eine Klasse aus Q als toprank aus.*

Die Klassifikatoren waren bisher diskrete Funktionen auf $\{0,1\}$. Bei *Weighted Voting* sind die Stimmen, wie man aus dem Namen entnehmen kann, gewichtet. Diese Gewichtung basiert z. B. auf der Fehlerabschätzung der Vorhersage vom verwendeten Lernverfahren. Für *Weighted Voting* gelten dieselben Definitionen wie bei *Voting* bis auf den Unterschied, dass es sich nun um reelle Funktionen handelt.

3.1.2 Voting Variante: Vote Against

Cutzu geht in [5] von einer neuen Interpretation des *Voting* aus. Anstatt dass jeder Klassifikator *für* die gewonnene Klasse stimmt, sollte *gegen* die verlorene Klasse gestimmt werden. Für einen binären Klassifizierer $K_{i,j}$, der auf eine Testinstanz angewendet, als Klassifikationsergebnis i bestimmt, impliziere eher $j \neq$ top rank, als dass $i =$ top rank gelte.

Definition 3.4 (Voting-against) *Sei $K_{i,j} : T \rightarrow [0,1]$ der binäre Klassifikator für die Klassen k_i, k_j und t die zu klassifizierende Testinstanz.*

$$top\ rank := arg\min_{i \in K} \sum_{j \in K} (1 - K_{i,j}(t))$$

Cutzu erläutert, dass eine Testinstanz x, deren korrekte *top rank* Klasse weder k_i noch k_j sei, könne bei der bisherigen *Voting* Methode zu *falsch positiven* Fehlern führen, indem fälschlicherweise $x \notin k_i \bigcup k_j$ der Klasse k_i oder k_j zugeordnet wird.

Auf den ersten Blick scheint diese neue Interpretation des *Voting* keine Vorteile zu bringen. In der Tat ist bei einem kompletten *Voting-against* und *Voting-for* das Ergebnis identisch.

$$arg\max_{i \in K} \sum_{j \in K} K_{i,j}(t) = arg\min_{i \in K} \sum_{j \in K} (1 - K_{i,j}(t))$$

Jedoch bietet es Vorteile, wenn nur eine Teilmenge der paarweisen Klassifikatoren ausgewertet wird. Unter der Annahme, dass die paarweisen Klassifizierer (fast) keine *falsch negativen* Fehler hervorrufen, ist *Voting-Against* dem *Voting-For* überlegen.

Voting-Against wird (fast) nie eine Stimme *gegen* die korrekte *top rank* Klasse abgeben und als Ergebnis wäre im schlimmsten Fall, dass die Menge $L := arg\min_{i \in K} \sum_{j \in K} (1 - K_{i,j}(t))$ mehrdeutig ist. Jedoch ist garantiert, dass *top rank* in L enthalten ist[1]. Bei *Voting-For* hingegen, ist dies nicht gewährleistet.

Zusammengefasst ist im Bezug auf eine Effizienzsteigerung mittels Reduktion der Vergleiche die *Voting-Against* Methode dem *Voting-For* vorzuziehen.

3.1.3 Decision Directed Acyclic Graph (DDAG)

Im Hinblick auf die Frage, welche Teilmenge der paarweisen Klassifizierer ausreicht, um mit der *Voting-Against* Methode ein akzeptables Klassifikationsergebnis zu erreichen, ist der DDAG-Algorithmus von Platt et. al. [18] interessant.

[1] Dies gilt unter der Annahme, dass die paarweisen Klassifizierer keine *falsch negativen* Fehler hervorrufen.

DDAG ist ein gerichteter azyklischer Graph, indem alle paarweisen Klassifizierer als Knoten definiert sind. Die Dekodierung stellt sich als Pfad von der Wurzel bis zum Blatt dar. Dabei ist hervorzuheben, dass bei einem n-Klassenproblem nur $n-1$ Vergleiche benötigt werden.

Der *DDAG*-Algorithmus geht von derselben Idee wie *Voting-Against* aus. Dem Autor ist unklar, ob diese Interpretation von Cutzu und Platt unabhängig zustande gekommen ist, jedoch kann gesagt werden, dass der Artikel von Platt 3 Jahre zuvor erschienen ist.

In Abbildung 3.1 ist die Herleitung geometrisch dargestellt.

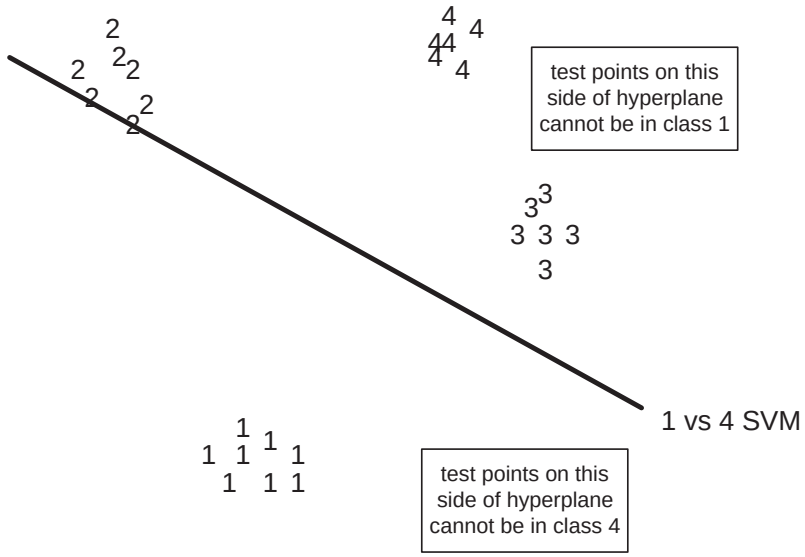


Abbildung 3.1: Voting-Against: Der paarweise Klassifikator von den Klassen 1 und 4 (Quelle: [18]).

Liegt der relevante Testpunkt z. B. auf der Seite zu Klasse 4, kann nicht automatisch gefolgert werden, dass die korrekte Klasse 4 ist (Klasse 2 und 3 sind möglich). Stattdessen erscheint die Interpretation, dass die korrekte Klasse *nicht* 1 ist, plausibler.

Definition 3.5 (Decision Directed Acyclic Graph (DDAG)) *Sei N die Menge der paarweisen Klassifizierer. Ein Decision Directed Acyclic Graph (DDAG) ist ein Entscheidungsbaum mit einem beliebigen aber festen Klassifizierer $K_{i,j} \in N$ als Wurzel. Die Wurzel und die Knoten haben stets zwei Nachfolger. Nämlich $K_{i,k}$ und $K_{l,j}$ mit $k_k \in K \backslash \{k_i, k_j\}, k_l \in K \backslash \{k_i, k_j\}$. Dabei sind die Klassifizierer dreieckartig angeordnet, sodass jeder Klassifizierer genau einmal vorkommt und die Gesamtanzahl der Knoten $\frac{n(n-1)}{2}$ ist.*

Die etwas grobe Definition im Hinblick auf die dreieckartige Anordnung wird einem schnell klarer, wenn man die Beispielabbildung 3.2 betrachtet.

Definition 3.6 (DDAG Transitionsfunktion) *Sei $t_{i,j} \in T$ der aktuelle Knoten und für die paarweisen Klassifikatoren gelte $K_{i,j} : T \to \{0,1\}$, dann ist die Transitionsfunktion $f_t : T \to T$ wie folgt definiert:*

$$T(t_{i,j}) = \begin{cases} t_{i,k} & \text{falls } K_{i,j}(t) = 1, \text{ also nicht } j, \\ t_{l,j} & \text{ansonsten, also nicht } i \end{cases}$$

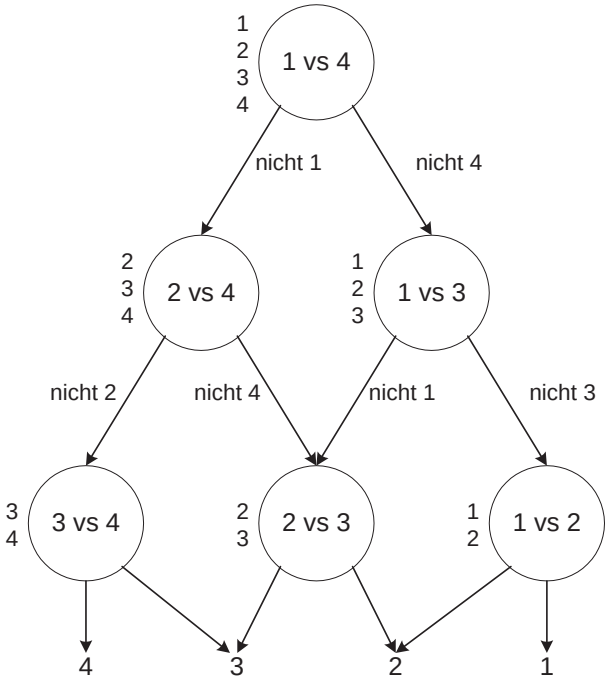


Abbildung 3.2: DDAG: 4 Klassen $K = \{1,2,3,4\}$. Die Dekodierung ist ein Pfad von der Wurzel bis zum Blatt. Die Blätter stellen das Klassifikationsergebnis dar (Quelle: [18]).

wobei $t_{i,k}$ und $t_{l,j}$ die direkten Nachfolger von $t_{i,j}$ sind.

Die Evaluierung des Entscheidungsbaumes mithilfe der Transitionsfunktion stellt die Dekodierung dar.

Da die Laufzeit des Algorithmus linear ist, sind die guten Ergebnisse in [18] nicht verwunderlich. Verglichen mit dem kompletten *Voting* war es 1.6 bis 2.3-mal schneller. Auf den getesteten Datensätzen (UCI Letter, UCI Covertype, USPS) mit SVM als Basislerner wurden daneben gute bis sehr gute Genauigkeiten ermittelt.

Der Hauptnachteil am *DDAG*-Algorithmus ist, dass sie von sehr starken Klassifikatoren ausgeht. Bei schwachen Klassifikatoren kann es vorkommen, dass die korrekte *top rank* Klasse k_i gegen eine andere Klasse k_j im paarweisen Vergleich verlieren kann. Befindet sich nun der paarweise Klassifikator $K_{i,j}$ bei der Ausführung des Algorithmus auf dem aktuell verfolgten Pfad, so wird k_i von der möglichen *top rank* Menge entfernt und kann somit nicht mehr als Klassifikationsergebnis bestimmt werden. In anderen Worten, die geringe Anzahl an Vergleichen $(n-1)$ wird durch den Verlust von redundanten Informationen, die bei schwächeren Klassifikatoren zur Steigerung der Genauigkeit helfen würden, ermöglicht.

Es sollte noch erwähnt werden, dass die Implementation des *DDAG* Algorithmus sehr simpel ist. Beginnend mit einer Liste P aller Klassen, werden zwei beliebige Klassen k_i, k_j ausgewählt, der entsprechende Klassifikator $K_{i,j}$ ausgewertet und die verlorene Klasse aus der Liste P entfernt. Dieser Vorgang wird so oft wiederholt bis die Liste P nur noch eine Klasse - das Klassifikationsergebnis - enthält.

Es existieren Situationen, in denen unklassifizierbare Regionen [15] entstehen können (siehe Abbildung 3.3). Eine Erweiterung zu DDAG von Takahashi und Abe [23] behandelt solche Probleme, indem Klassenpaare mit höherem Potenzial zur Verallgemeinerung in die oberen Knoten des Graphen platziert

werden.

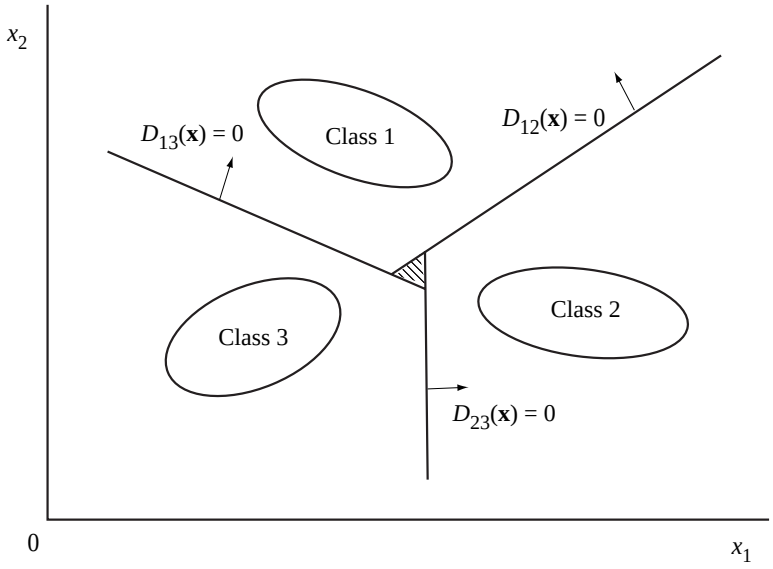


Abbildung 3.3: Unklassifizierbare Regionen bei paarweisen Vergleichen (Quelle: [23]).

3.2 Quick Weighted Voting

Die Komplexität des *DDAG*-Algorithmus ist zwar linear in Anzahl der Klassen und somit bedeutend schneller als die übliche *Weighted Voting* Dekodierung, dafür verliert es aber an Genauigkeit. Es wäre wünschenswert einen Kompromiss zwischen Genauigkeit und Effizienz herzustellen. Die Beobachtungen des Position-Errors aus den Ergebnissen des Schweizer-Systems (siehe 4.5.1), die darauf hindeuteten, dass für die Klassifikation nicht alle Vergleiche notwendig sind, waren Anlass, um zu untersuchen, ob eine effizientere Klassifikationsphase möglich ist.

Dabei entstand ein Algorithmus, der auf Weighted Voting basiert und eine Komplexität von $n \cdot \log(n)$ aufweist.

Aus dem Sport kennt man Situationen in denen ein Spieler uneinholbar für die anderen Gegner ist. Übertragen auf das Klassifikationsproblem existieren also Situationen, in denen keine zusätzlichen Vergleiche mehr nötig sind, um den top rank zu bestimmen. Gerade diese einfache Tatsache wird im Quick Weighted Voting ausgenutzt. Es ist klar, dass eine geschickte Reihenfolge diese Situationen begünstigen, nämlich wenn man so oft wie möglich mit der korrekten top rank Klasse spielt. Diese einfache Überlegung war die Hauptidee des QWeighted. Anhand eines konkreten Beispiels soll dieser Sachverhalt nochmal deutlich gemacht werden.

3.2.1 Ein kleines Beispiel

Aus aktuellem Anlass wurde ein Beispiel aus der FIFA Fußball-WM Deutschland 2006^{TM} benutzt. Die Tabelle 3.1 gibt den Punktestand[2] der Gruppe F nach allen Vorrundenspielen wieder.

Tabelle 3.1: Spielergebnisse aus der Gruppe F in der WM 2006

12 Jun	AUS : JPN	3:1
13 Jun	BRA : CRO	1:0
18 Jun	BRA : AUS	2:0
18 Jun	JPN : CRO	0:0
22 Jun	JPN : BRA	1:4
22 Jun	CRO : AUS	2:2

Pos	Land	Pkt
1.	Brasilien	9
2.	Australien	4
3.	Kroatien	2
4.	Japan	1

Tabelle 3.2: Punktestand nach den ersten fünf Spielen

Pos	Land	Pkt	gespielt
1.	Brasilien	9	(3/3)
	Australien	3	(2/3)
	Kroatien	1	(2/3)
	Japan	1	(3/3)

Betrachtet man den Punktestand nach den ersten fünf Spielen (Tabelle 3.2), ist klar, dass Brasilien (BRA) Gruppenerster wird, unabhängig davon, wie die anderen Länder spielen würden. Übertragen auf die Klassifikation kann man den letzten Vergleich auslassen, da es bei der Klassifikation nur um die Bestimmung der top rank Klasse geht.

Anhand dieses Beispiels wird auch deutlich, dass eine geschickte Reihenfolge die Anzahl der vernachlässigbaren Vergleiche vergrößern kann. Wenn nämlich Brasilien gleich am Anfang dreimal hintereinander gespielt hätte (Tabelle 3.3),

[2] Zur Berechnung der Punkte: 3 - Sieg, 1 - Unentschieden, 0 - Verloren

Tabelle 3.3: Punktestand nach den Spielen BRA:CRO, BRA:AUS und BRA:JPN

Pos	Land	Pkt	gespielt	max Pkt. möglich
1.	Brasilien	9	(3/3)	-
	Australien	0	(1/3)	6
	Kroatien	0	(1/3)	6
	Japan	0	(1/3)	6

wären keine weiteren Vergleiche mehr nötig gewesen. Damit wären bereits nach 3 Paarungen der korrekte top rank ermittelt.

3.2.2 Theoretische Überlegungen

Zunächst soll die Problemstellung formalisiert werden, dazu werden einige Variablen deklariert:

- die Menge der *Klassen* $K = \{k_1, k_2, k_3, \ldots, k_n\}$
- für jede Klasse k_i:
  - die bisherige *Anzahl gespielter Paarungen* $\tilde{p}_i$
  - die bisherige *Summe der gewichteten Stimmen* $\tilde{W}_i$

Es wird ein sogenannter Limit $\tilde{L}_i$ für jede Klasse k_i eingeführt, das die minimale Differenz zu jedem Zeitpunkt zum perfekten Voting $W_i = n-1$ darstellt.

Definition 3.7 (Limit) *Das Limit $\tilde{L}_i$ bezüglich einer Klasse i ist wie folgt definiert:*

$$\tilde{L}_i := \tilde{p}_i - \tilde{W}_i$$

Die Tilde „˜“ über den Variablen soll darauf hinweisen, dass diese Variablen nicht den entsprechenden Werten nach einem kompletten Weighted Voting darstellen, sondern den Wertestand in einem bestimmten Zustand. Da der konkrete Spielzustand im Folgenden nicht von Bedeutung ist, wurde keine explizite Indizierung vorgenommen. Darüber hinaus wurde die Verbindung zu einer konkreten Testinstanz t vernachlässigt.

Für folgende Aussagen wird angenommen, dass der top rank eindeutig sei. Auf den mehrdeutigen Fall wird am Ende des Kapitels eingegangen.

Lemma 3.2.1 *Annahme: Der Top Rank bezüglich Weighted Voting ist eindeutig, d. h.:*

$$W_{top} > W_i, \forall i \in \{1 \ldots n\} \backslash \{top\}$$

Für das Limit bedeutet das:

$$L_{top} < L_i, \forall i \in \{1 \ldots n\} \backslash \{top\}$$

Nachfolgend sind weitere Aussagen, die später hilfreich sein werden.

Lemma 3.2.2 *Seien $k_i, k_j \in K$ zwei beliebige Klassen mit $k_i \neq k_j$ und $\tilde{L}_i$ das Limit vor dem Vergleich (k_i, k_j) und $\tilde{L}_i^+$ nach dem Vergleich, so gilt:*

$$\tilde{L}_i \leq \tilde{L}_i^+ \text{ und analog } \tilde{L}_j \leq \tilde{L}_j^+$$

Beweis 3.2.1 *Es gilt:* $\tilde{L}_i = \tilde{p}_i - \tilde{W}_i$ *und* $\tilde{L}_i^+ = \tilde{p}_i^+ - \tilde{W}_i^+$. *Weiter gilt* $\tilde{p}_i^+ = \tilde{p}_i + 1$ *und* $\tilde{W}_i^+ = \tilde{W}_i + x$, *wobei* $x \in [0,1]$. *Da* $x \leq 1$, *folgt* $\tilde{p}_i - \tilde{W}_i \leq \tilde{p}_i - \tilde{w}_i + (1-x) = (\tilde{p}_i + 1) - (\tilde{W}_i + x) = \tilde{p}_i^+ - \tilde{W}_i^+$ *und somit* $\tilde{L}_i \leq \tilde{L}_i^+$.

Lemma 3.2.3 *Zu jedem Zeitpunkt im Spielverlauf ist das aktuell beste Limit* $\tilde{L}_{best} := \min(\tilde{L}_i), \forall i \in K$ *kleiner oder gleich dem korrekten Limit von* k_{top}*:*

$$\tilde{L}_{best} \leq L_{top}$$

Beweis 3.2.2 *Für das Limit der top rank Klasse gilt laut 3.2.2* $\tilde{L}_{top} \leq L_{top}$, *gleichgültig wieviele Vergleiche* k_{top} *bisher durchgeführt hat. Und da per Definition* $\tilde{L}_{best} \leq \tilde{L}_{top}$ *gilt, folgt die Behauptung.*

Lemma 3.2.4 *Ist der tatsächliche Limit für den Top Rank* L_{top} *gegeben, dann benötigt man höchstens* $n-1$ *Vergleiche, um zu verifizieren, ob für* **eine** *Klasse* k_i *gilt* $k_{top} = k_i$.

Beweis 3.2.3 *Fallunterscheidung, in beiden Fällen werden nur Vergleiche mit der Klasse* k_i *ausgewertet:*

- *Für den Fall* $i =$ *top rank ist klar, dass man alle Vergleiche von Klasse i, also genau* $n-1$ *auswerten muss, um die Gleichheit zu verifizieren.*
- *Für* $i \neq$ *top rank gilt* $L_{top} < L_i$. *Da das Limit* $\tilde{L}_i$ *nach jedem Vergleich bestenfalls gleich bleiben kann (3.2.2), ist es möglich, dass die Anzahl der benötigten Vergleiche kleiner als* $n-1$ *sein kann. Sie ist jedoch nach oben mit* $n-1$ *beschränkt.*

Die Aussage aus 3.2.2 besagt, dass die Limits sich nur verschlechtern können. Nach einer bestimmten Anzahl a von Spielen wird $\tilde{L}_{\text{best}} = L_{\text{top}}$ gelten und mit dem Lemma 3.2.4 werden somit höchstens noch $n-1$ Vergleiche[3] zusätzlich notwendig sein. Die Frage ist, ob es eine Möglichkeit gibt a zu minimieren.

Ohne weiteres Hintergrundwissen erscheint die Heuristik, immer mit der Klasse mit dem aktuell besten Limit zu spielen, die Günstigste zu sein.

Zusammengefasst ergibt sich folgender Algorithmus:

Algorithmus 2 : Quick Weighted Voting

```
1  begin
2      while top rank nicht ermittelt do
3          Spieler_1 ← wähle k_i mit minimalen L̃_i
4          Spieler_2 ← wähle k_j mit minimalen L̃_j und k_j ≠ k_i und Paarung
           (k_i,k_j) noch nicht gespielt
5          if Es existiert kein Spieler_2 then
6              k_top ← Spieler_1
7          else
8              Spiele(Spieler_1, Spieler_2)
9              Aktualisiere Statistiken
10 end
```

[3] Oft kann die obere Schranke geringer sein als $n-1$, da zuvor bereits einige Vergleiche mit der korrekten *top rank* Klasse stattgefunden haben können.

Für die Paarungsbildung geht der Algorithmus folgendermaßen vor. Zunächst wird der Spieler mit dem aktuell besten Limit als Spieler_1 ausgewählt. Daraufhin wird ein Gegenspieler gesucht, dessen Limit minimal unter den restlichen Spielern ist und bisher noch nicht gegen Spieler_2 gespielt hatte. Sollte der Spieler mit dem minimalen Limit aus der Restmenge der Klassen $K \backslash \{\text{Spieler}_1\}$ schon gegen Spieler_1 gespielt haben, so wird der Spieler mit dem zweitbesten Limit überprüft. Für den Fall, dass kein Gegenspieler gefunden werden konnte, gilt, dass Spieler_1 bereits gegen alle möglichen Spielern gespielt hat und damit das Limit von Spieler_1 als L_{top} verifiziert wurde und dass $\text{Spieler}_1 = k_{top}$ gilt. Ansonsten wird die Paarung durchgeführt und die Statistiken aktualisiert.

Folgende Aussage soll noch einmal unterstreichen, dass das Ergebnis von *Quick Weighted Voting* mit dem eines kompletten *Weighted Voting* identisch ist.

Lemma 3.2.5 (Quick Weighted Voting Ergebnis) *Für die Klasse $k_p \in K$ mit dem verifizierten minimalen Limit L_p für die Testinstanz t gilt :*

$$k_p = arg \max_{k_i \in K} \sum_{k_j \in K} K_{i,j}(t)$$

Beweis 3.2.4

$$\begin{aligned}
L_p &= \min_{k_i \in K} L_i \\
k_p &= arg \min_{k_i \in K} L_i \\
&= arg \min_{k_i \in K} (p_i - W_i) \\
&= arg \min_{k_i \in K} (p_i - \sum_{k_j \in K} K_{i,j}(t)) \\
&= arg \max_{k_i \in K} (\sum_{k_j \in K} K_{i,j}(t) - p_i) \\
&= arg \max_{k_i \in K} \sum_{k_j \in K} K_{i,j}(t)
\end{aligned}$$

Da also die Klasse mit dem minimalen Limit die korrekte *top rank* Klasse nach einem kompletten Weighted Voting ist, muss nur noch gezeigt werden, dass der Algorithmus diese Klasse nun auch bestimmt, also korrekt arbeitet.

3.2.3 Korrektheit

Am Anfang des Algorithmus gilt: $\tilde{L}_i = 0 \ \forall k_i \in K$. Das heißt, es werden zunächst zufällig zwei Spieler k_i, k_j ausgewählt. Es findet ein paarweiser Vergleich statt und die veränderten Statistiken $(\tilde{L}_i, \tilde{L}_j, \tilde{p}_i, \tilde{p}_j, \tilde{W}_i, \tilde{W}_j)$ werden aktualisiert. Nach jedem Durchlauf füllt sich die Limittabelle und die Heuristik gewinnt an Bedeutung. Der Algorithmus bricht ab, sobald für die aktuell beste Klasse $\tilde{k}_{\text{best}}$ keine ungespielte Gegnerklasse mehr existiert. Da laut Aussage 3.2.2 keine Klasse existiert, die die Klasse k_{best} einholen kann, sind weitere Auswertungen unnötig. Somit gilt $\tilde{k}_{\text{best}} = k_{\text{best}}$ und das Limit L_{best} ist somit als $L_{\text{top rank}}$ verifiziert.

Für den Fall, dass die Annahme der Eindeutigkeit des *top rank* nicht gilt, sind weitere Auswertungen nötig. Angenommen k_i sei durch obigen Algorithmus als *top rank* ermittelt worden. Klasse k_j habe dasselbe (vorläufige) Limit wie

k_i, also $\tilde{L}_j = L_i$. Um Klasse k_j nun wirklich zu der *top rank* Menge zu zählen, muss aber $L_j = L_i = L_{\text{best}}$ gelten. dass heißt, die restliche Menge der zu k_j inzidenten Klassifikatoren müssen ausgewertet werden, bis entweder die Gleichheit bestätigt wird oder es wird vorher abgebrochen, wenn $\tilde{L}_j$ sich einmal verschlechtert, sodass $L_j = L_i$ nicht erreicht werden kann.

3.2.4 Komplexität

Bis hierhin ist noch nicht ersichtlich, wieso der Algorithmus eine durchschnittliche Laufzeit von $n \cdot \log(n)$ aufweisen sollte. Darauf soll nun eingegangen werden.

Zunächst sollen kurz die beiden extremen Fälle Best- und Worst-Case betrachtet werden.

Es ist ersichtlich, dass der Algorithmus nie mehr Vergleiche als *Weighted Voting* benötigt. Im Worst-Case Fall ist die Anzahl der Vergleiche identisch mit einem kompletten Weighted Voting ($\frac{n(n-1)}{2}$ Vergleiche). Ein solches Szenario wäre beispielsweise eine gleichmäßige Verteilung aller Stimmen auf die Klassen.

Beispiel 3.1 (Ein Worst-Case Szenario) *Das Problem bestehe aus n Klassen und alle Vergleiche (k_i, k_j) ergäbe das Ergebnis $w_{i,j} = 0.5 = w_{j,i}$. Das bedeutet, dass alle Klassen mit der gleichen Anzahl an gespielten Paarungen dasselbe Limit besitzen.*

$$\forall k_i, k_j \in K . \tilde{p}_i = \tilde{p}_j \Leftrightarrow \tilde{L}_i = \tilde{L}_j$$

Weiter gilt für beliebige Klassen $k_m, k_n \in K$:

$$\tilde{p}_m > \tilde{p}_n \Leftrightarrow L_i > L_j$$

Das wiederum bedeutet, dass die Heuristik des Algorithmus darauf reduziert wird, dass immer die Klassen mit der kleinsten Anzahl von Spielen in den Schritten 2 und 3 bevorzugt ausgewählt werden. Somit spielt jede Klasse genau $n-1$ mal, bevor der Algorithmus abbricht. Insgesamt ergäbe das $\frac{n(n-1)}{2}$ Vergleiche.

Es muss erwähnt werden, dass man im obigen Beispiel von keiner Eindeutigkeit des *top rank's* ausging. Auf der anderen Seite ist es theoretisch möglich im *Best-Case* Fall mit dem Algorithmus eine Laufzeit von $n-1$ zu erreichen.

Beispiel 3.2 (Ein Best-Case Szenario) *Das Problem bestehe wieder aus n Klassen und die korrekte* top rank *Klasse $k_{\text{top rank}}$ gewinne immer perfekt, d. h. für das Limit gelte $L_{\text{top rank}} = 0$. Weitere Voraussetzung ist, dass $k_{\text{top rank}}$ in der ersten Iteration des Algorithmus als $Spieler_1$ ausgewählt wird. Übereinstimmend mit Satz 3.2.4 werden nun genau $n-1$ Vergleiche benötigt, um zu verifizieren, dass $k_{\text{top rank}}$ die korrekte* top rank *Klasse ist.*

Es folgt der durchschnittliche Fall. Die $n \cdot \log(n)$ Komplexität scheint auf den ersten Blick nicht ohne stark einschränkende Annahmen nachvollziehbar. In der Tat beruht diese Aussage auf eine Annahme, die aber anhand empirischer Auswertungen von realen Datensätzen durchaus für die Praxis angenommen werden kann.

Zunächst soll noch einmal kurz auf den *DDAG*-Algorithmus zurückgeblickt werden. Es wurde erwähnt, dass man sich die konkrete Anwendung folgendermaßen vorstellen kann. Es werden zwei zufällige Klassen $k_i, k_j \in K$ ausgewählt und nach der Klassifikatorauswertung $K_{i,j}$ auf der Dateninstanz, die verlorene Klasse aus der Klassenmenge entfernt, z. B. sei dies Klasse k_j. Im nächsten Schritt wird ein neuer Gegner für k_i aus der reduzierten Klassenmenge bestimmt und evaluiert. Dieses Schema wird solange wiederholt, bis die Klassenmenge nur noch aus einer Klasse besteht. Somit sieht man sofort, dass der Algorithmus immer den korrekten top rank bestimmen kann, wenn die Annahme gelten würde, dass die top rank Klasse in allen Spielen gewinnt.

Im Grunde lockert der Quick Weighted Algorithmus diese Annahme zugunsten der Genauigkeit. Wie man aus den *best-* und *Worst-Case* Szenarien erahnen kann, wird die Laufzeit von der *Verteilung der (gewichteten) Stimmen*, insbesondere von dem *top rank*, beeinflusst. Für die Betrachtung des durchschnittlichen Falles wird versucht, eine Beziehung zwischen Laufzeit und Verteilung herzustellen. Der Versuch eine formale Komplexitätsbestimmung zu entwickeln ist leider nicht geglückt, jedoch sind aus Gründen der Vollständigkeit einige informelle theoretische Überlegungen hierzu im folgenden Text beschrieben.

Für die Ermittlung der durchschnittlichen Laufzeit soll eine andere Interpretation des Algorithmus betrachtet werden. Während des Spielverlaufs ist k_{best} nicht eindeutig, d. h. es existiert eine Teilmenge von Klassen $K' \subseteq K$, die alle das gleiche Limit L_{best} besitzen: $\forall k_i \in K' \leftrightarrow L_i = L_{best}$. Ein Vergleich zweier Klassen k_i, k_j verändert mindestens einer ihrer Limits $\tilde{L}_i, \tilde{L}_j$. Für die Heuristik des Algorithmus sind alle Klassen $k \in K'$ gleichwertig, in dem Sinne, dass als $Spieler_1$ eine zufällige Klasse aus K' ausgewählt wird. Dies soll kurz die andere Sichtweise andeuten.

Zu Beginn des Algorithmus sind alle Limits mit null initialisiert: $L_i = 0, \forall i \in \{1 \dots n\}$. Es gilt also $K' = K$.

Lemma 3.2.6 *Jede Klassifikatorauswertung $K_{i,j}$ für zwei beliebige Klassen $k_i, k_j \in K$ führt zu einer Verschlechterung des Limits für mindestens eine der Klassen k_i, k_j.*

Lemma 3.2.7 *Nach mindestens $\lceil \frac{n}{2} \rceil$ Vergleichen ist die Menge K' um mindestens die Hälfte reduziert.*

Beweis 3.2.5 *Die Anzahl der Klassen in K' ist zu Beginn n. In jedem der $\lceil \frac{n}{2} \rceil$ Vergleiche wird das Limit für mindestens eine der beiden Klassen größer. Dies führt zu einer Reduzierung um mindestens einer Klasse pro Vergleich.*

Eine ähnliche Aussage zur obigen ist:

Lemma 3.2.8 *Nach maximal $\lceil \frac{n}{2} \rceil$ Vergleichen hat jede Klasse mindestens 1-mal gespielt*

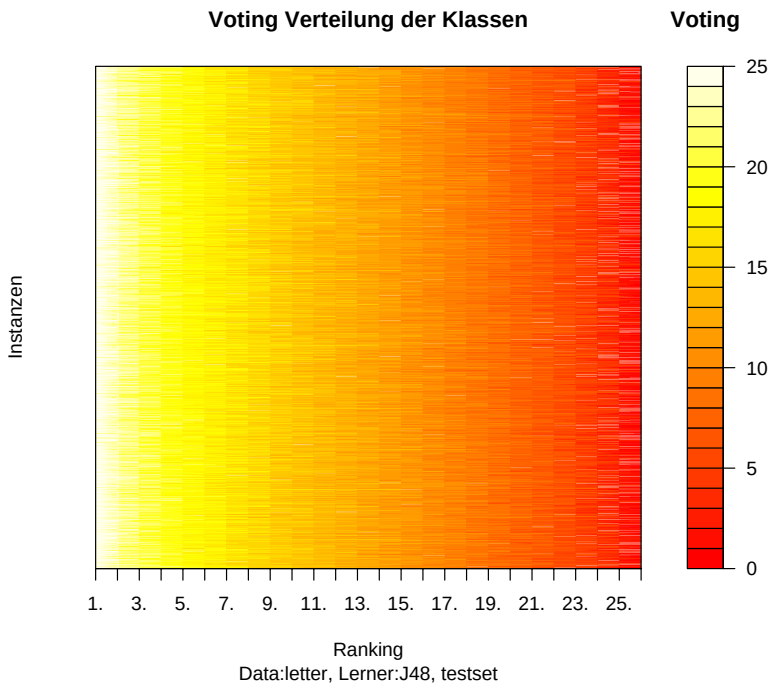
3.2.5 (Weighted) Voting Verteilungen

Es konnte zwar nicht formal gezeigt werden, dass die durchschnittliche Komplexität $n \cdot \log(n)$ ist, jedoch wurde versucht, die Vermutungen über den Voting Verteilungen, die sich wahrscheinlich günstig auf die Effizienz des Algorithmus auswirken, mit Auswertungen von realen Datensätzen zu bestätigen.

Ähnlich wie im Zusammenhang zum DDAG-Algorithmus ist sicherlich eine möglichst perfekte Siegesbilanz des top rank einer der wesentlichen Eigenschaften, die die Laufzeit des Algorithmus begünstigen. Daneben wäre für die restlichen Klassen eine Punkteverteilung von Vorteil, die sich klar abgrenzt. Genauer betrachtet ist dies jedoch nicht als allgemeingültig zu betrachten, da letztendlich einzelne Klassifikatorergebnisse und die Art in welcher Reihenfolge sie angeordnet sind, großen Einfluss auf die Laufzeit des Algorithmus haben.

Anhand einiger Datensätze soll grafisch die Verteilung der Stimmen nach einem kompletten Voting bzw. Weighted Voting dargestellt werden.

Abbildung 3.4: Voting Verteilung für `letter`



Die vertikale Achse stellt die Instanzen dar. Dabei stellt jede einzelne Linie eine Instanz aus dem Datensatz dar. Die Ranking Platzierungen sind horizontal verteilt. Von links nach Rechts, beginnend vom *top rank* absteigend zum *bottom rank*. Die Farben stellen die Summe der gewichteten Stimmen dar. In allen Graphen ist von links nach rechts gesehen ein Farbverlauf von Weiß nach Rot zu sehen. Dies stellt nur den Zusammenhang dar, dass je schlechter die Platzierung ist (horizontal, rechts), desto rötlicher ist die Farbe für alle Testinstanzen (vertikal).

Viel wichtiger ist, dass wenn man die Summe der Stimmen für den *top rank* betrachtet, alle Testinstanzen der Datensätze nahezu perfekt sind (weiß). Das heißt durch die Grafiken lässt sich einfach die Annahme bei *QWeighted* bestätigen.

Die Beispiele zum *Worst-* und *Best-Case* in dem vorigen Kapitel lassen vermuten, dass je gleichmäßiger die Verteilung der Stimmen ist, desto schlechter ist die Laufzeit von *QWeighted*. Das Worst-Case Szenario wie es in Beispiel 3.1 konstruiert wurde, wäre in dieser Form grafisch betrachtet einfarbig.

Diese Grafiken wurden unter keinem bestimmten Kriterium für einige Datensätze erstellt. Sie stellen recht simpel die Charakteristik der Verteilung von Stimmen auf realen Datensätzen dar.

Abbildung 3.5: Voting Verteilung für `vowel` (Basislerner: JRip)

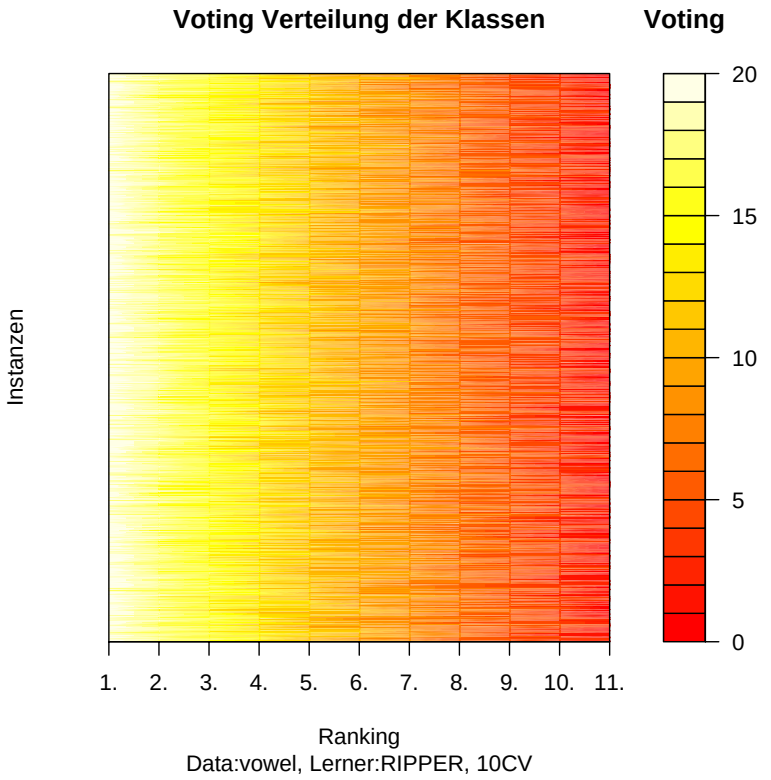
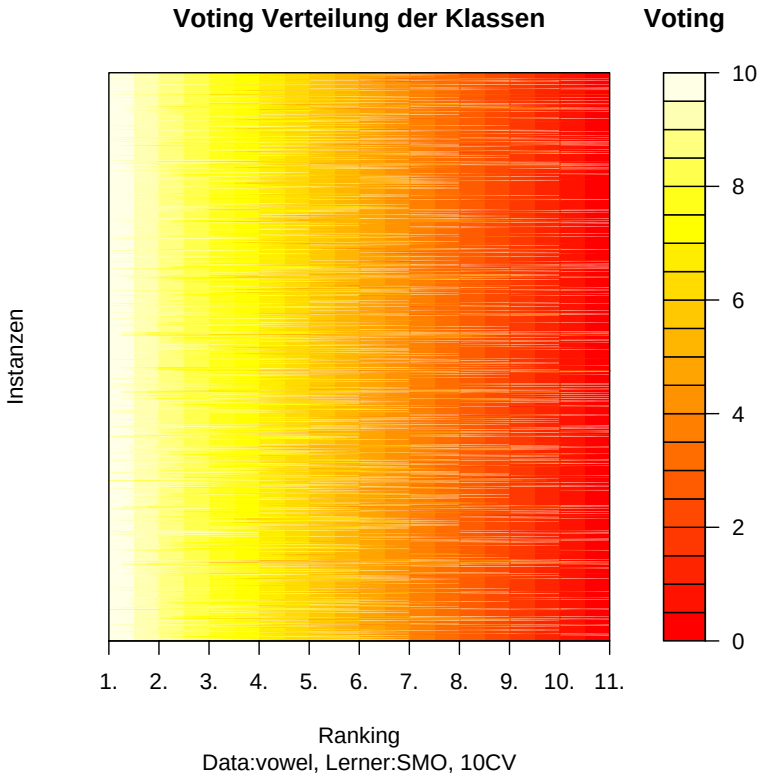


Abbildung 3.6: Voting Verteilung für `vowel` (Basislerner: SMO)



3.2.6 Auswertungen

WEKA Framework

Alle Implementationen und Auswertungen wurden innerhalb des WEKA-Framework durchgeführt. WEKA (Waikato Environment for Knowledge Analysis) [25] ist eine in JAVA entwickelte Umgebung, die speziell für Machine Learning und Data Mining entwickelt wurde. Sie stellt eine Menge an Lernalgorithmen zur Verfügung sowie unter anderem Tools für Daten Pre-processing, Klassifikation, Regression und Clustering. Zum Zeitpunkt dieser Diplomarbeit ist die aktuellste Development WEKA Version 3.5.2.

Im Rahmen dieser Diplomarbeit wurde die bereits bestehende Klasse `MultiClassClassifier.java` um *QWeighted* erweitert.

Datensätze

Alle verwendeten Datensätze stammen aus den öffentlich verfügbaren Datensammlungen der UCI Repository[17]. Die UCI Repository stellen gesammelte Datensätze zur Verfügung, die sich für Machine Learning oder Data Mining Problemstellungen eignen.

Es wurden zufällig mehrere Datensätze (siehe Tabelle 3.4) ausgesucht, die als einzige Bedingung hatten, mehr als 2 Klassen zu beinhalten.

Tabelle 3.4: Verwendete Datensätze. Die Spalten 2 und 3 beinhalten die Instanzenanzahl der Trainings- bzw. Testdaten. Die restlichen drei Spalten zeigen die Anzahl der symbolischen, numerischen Attribute und Anzahl der Klassen.

Name	Training	Test	sym.	num.	Klassen
vehicle	846	-	0	18	4
glass	214	-	0	9	7
image	2310	-	0	19	7
yeast	1484	-	0	8	10
vowel	528	462	0	10	11
soybean	683	-	35	0	19
letter	16000	4000	0	16	26

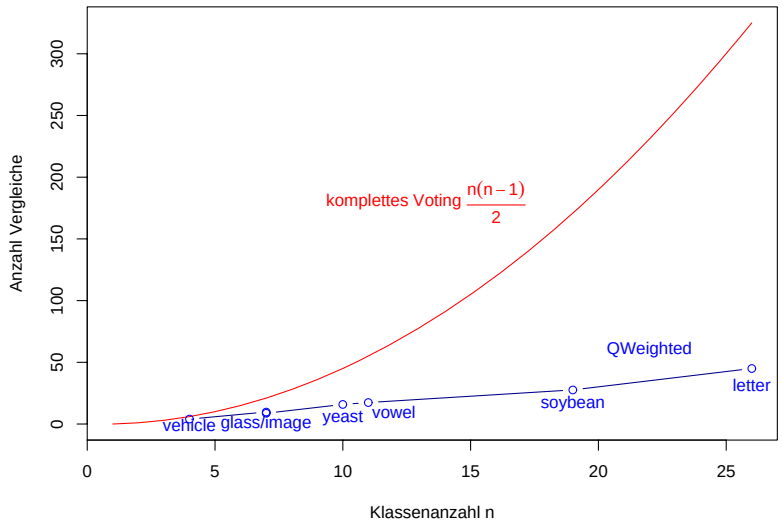
Art der Analyse

Da die verwendeten Basislerner nicht alle Klassen-symmetrisch sind, wurden alle Auswertungen mit *Double Round-Robin* ausgewertet. Es soll beachtet werden, dass die entstehenden Ergebnisse bezüglich der Anzahl der Vergleiche alle halbiert wurden. Das heißt, es werden für jedes beliebige Klassenpaar (k_i, k_j) die Klassifizierer $K_{i,j}$ und $K_{j,i}$ gelernt und die Evaluierung dieser beiden Klassifizierer wurde als ein Vergleich gezählt. Beide Klassifikatoren sind dabei gleichmäßig gewichtet, das heißt, die Ergebnisse der beiden Klassifikatoren werden aufsummiert und auf $[0,1]$ normiert. Diese Modifikation wurde insbesondere nicht nur beim *QWeighted*, sondern auch bei allen anderen ausgewerteten Verfahren angewendet.

Anzahl der Vergleiche: QWeighted im Vergleich zum kompletten Weighted Voting

In Tabelle 3.5 ist die ermittelte Laufzeit der Klassifikationsphase mit *QWeighted* zu sehen. Als Basislerner wurde dabei die *RIPPER* Implementation von WEKA (JRip) verwendet. Als Validierungsmethode wurde 10-fach Cross-Validation angewendet, außer bei dem Datensatz `letter`. Bei `letter` wurden die Testdaten herangezogen. Außerdem wurde bei dem Datensatz `vowel` die

Abbildung 3.7: QWeighted im Vergleich zu kompletten Voting



Trainingsmenge um die Testinstanzen erweitert, sodass die Instanzenzahl insgesamt 990 beträgt.

Wie man sehen kann, ist die Reduzierung der Vergleiche, je größer die Klassenanzahl des Datensatzes wird, stärker. Benötigt *QWeighted* noch bei dem Datensatz `vehicle` (4 Klassen) etwa 66% aller möglichen Vergleiche, reduziert es sich bei `yeast` (10 Klassen) schon auf 35%. Beim Datensatz `letter` mit der größten Klassenanzahl von 26 werden gar nur noch durchschnittlich 14% der Klassifikatoren ausgewertet.

Abbildung 3.7 verdeutlicht diese Tendenz grafisch. Das weist auf eine geschätzte Laufzeit von $n \cdot \log n$ hin.

Um diese Tendenz auch für größere n zu zeigen, fehlten leider die entsprechenden Datensätze. Der unter anderem verwendete Datensatz `letter` stellt mit 26 Klassen das größte Multiklassifikationsproblem dar, dass in der UCI Repository vorhanden ist. Da es schwer ist Aussagen über die Komplexität von Algorithmen auf kleinen n zu treffen, stieß man hier auf ein Problem. Eine Lösung wäre es, wie in [11], synthetisch generierte Datensätze zu verwenden. Dieser Ansatz wurde in dieser Diplomarbeit nicht weiter verfolgt.

Tabelle 3.5: Durchschnittliche Vergleichsanzahl von QWeighted - n ist die Anzahl der Klassen, $\frac{n(n-1)}{2}$ die Anzahl der Vergleiche von Weighted Voting und QWeighted die durchschn. Anzahl der Vergleiche von Quick Weighted Voting (Ripper, 10CV)

Datensatz	n	$\frac{n(n-1)}{2}$	QWeighted
vehicle	4	6	3,9844
glass	7	21	9,7476
image	7	21	8,7459
yeast	10	45	15,8749
vowel	11	55	17,4212
soybean	19	171	27,6510
letter	26	325	45,0053

QWeighted unter verschiedenen Basislernern

Daneben wurde *QWeighted* unter verschiedenen Basislernern ausgewertet. Bei allen Basislernern wurden keine weitergehenden Feineinstellungen vorgenommen. Sie wurden in der Standardeinstellung belassen. In Tabelle 3.6 sieht man,

dass die Anzahl der Vergleiche für alle Basislerner sich in derselben Größenklasse befinden. Dies macht deutlich, dass dieser Algorithmus auf der Metaebene aktiv ist, das heißt, dass es unabhängig vom verwendeten Basislerner ist - im Sinne von der Art des Lernens (Regellernen, SVM, Naive Bayes). Von einer präziseren Klassifikationsleistung der jeweiligen Basislerner wird es hingegen sicherlich profitieren.

Tabelle 3.6: Quick Weighted Voting unter verschiedenen Basislerner

Datensatz	JRip	NaiveBayes	C4.5(J48)	SVM
vehicle	3,9844	4,2660	3,9581	3,6442
glass	9,7476	9,5764	9,6896	9,9249
image	8,7459	9,0299	8,5472	8,2905
yeast	15,8749	15,8608	15,4774	15,5169
vowel	17,4212	17,0939	17,1253	15,2808
soybean	27,6510	27,6972	29,4460	28,3645
letter	45,0053	44,3960	47,7705	42,2618

3.2.7 Einordnung als Suchalgorithmus

Den Quick Weighted Voting Algorithmus kann man zu der Reihe von Greedy-Algorithmen zählen. Der folgende Überblick zu Greedy Algorithmen wurde hauptsächlich aus [4] entnommen.

Optimierungsalgorithmen werden typischerweise iterativ als Sequenz von Schritten abgearbeitet, wobei in jedem Schritt eine Menge von Entscheidungen zur Auswahl stehen. Das Greedy-Prinzip wählt dabei in jedem Schritt die, bezüglich einer Heuristik, „günstigste“ Entscheidung aus. Es wird sozusagen schrittweise das lokale Optimum verfolgt mit der Hoffnung auf diese Weise das globale Optimum zu erreichen. Diese lokalen Suchverfahren, wie das Gradientenverfahren, führen nicht für alle Probleme zu einem optimalen Ergebnis, jedoch stellen sie eine einfache effiziente Herangehensweise für viele Probleme dar.

Definition 3.8 (Greedy Prinzip) *Gegeben sei eine Menge E von Entscheidungen und sei o. b. d. A. $h(x) : E \rightarrow [0,1]$ die Heuristikfunktion, dann wird in jeder Situation einer Optimierungsphase, die Entscheidung $j \in E$ ausgewählt für die gilt:*

$$\forall i \in E \rightarrow h(j) \geq h(i)$$

.

Einige der prominenteren Vertreter des Greedy Prinzips sind der Huffman Code, Prim's Algorithmus und der Kruskal Algorithmus. Huffman Codes sind eine sehr effiziente Methode um Daten zu komprimieren. Die beiden letzteren Algorithmen stammen aus der Graphentheorie und lassen eine effiziente Berechnung des minimalen Spannbaums eines zusammenhängenden, ungerichteten, gewichteten Graphen zu. Auf den Kruskal Algorithmus wird näher eingegangen, da gewisse Analogien zum QWeighted Algorithmus bestehen.

Es werden zunächst einige benötigte Definitionen wiedergegeben, die aus [12] entnommen wurden:

Definition 3.9 (Definitionen zur Graphentheorie) *Ein Graph ist ein Paar $G = (V,E)$, wobei V eine beliebige Menge ist und E eine Menge zweielementiger Teilmengen von V, also $E \subseteq \{(v,w)|v,w \in V, v \neq w\}$. Die Elemente von V heißen* **Ecken** *oder* **Knoten** *und die Elemente von E* **Kanten** *von G*

- *Sei eine Kante $e = (v,w)$ gegeben. Man sagt e ist* **inzident** *mit v und w.*
- *Ein Graph $H = (W,F)$ mit $W \subseteq V$ und $F \subseteq E$ heißt* **Teilgraph** *von G. Gilt zusätzlich $F = E \bigcap \{(v,w) | v,w \in W\}$, dann heißt H* **Untergraph** *von G.*
- *Eine Folge $(v_0, v_1, ..., v_n)$ von Knoten von G heißt* **Kantenzug**, *wenn $e_i := (v_i, v_{i+1}) \in E$ gilt für $i = 0, 1, ..., n-1$. Im Fall $v_0 = v_n$ handelt es sich um einen* **geschlossenen** *und sonst um einen* **offenen** *Kantenzug. Sind alle Kanten verschieden, so spricht man im offenen Fall von einem* **Weg** *und in einem geschlossenen Fall von einem* **Kreis**. *Kommt außerdem jede Ecke v_j nur einmal vor, dann heißt der Weg ein einfacher Weg, analog für den Kreis.*
- *Zwei Ecken v und w sind* **verbindbar**, *wenn ein Kantenzug von v nach w existiert. Sind je zwei Knoten von G verbindbar, dann heißt G* **zusammenhängend**.
- *Ein Graph ist* **gerichtet**, *wenn die Menge E einer Ordnung unterliegt. $(v,w) \neq (w,v)$.*
- *Für jede Kante e heißt $w(e)$ das* **Gewicht** *von e.*

Definition 3.10 (Baum) *Ein azyklischer, zusammenhängender Graph wird* **Baum** *genannt.*

Definition 3.11 (minimaler Spannbaum eines Graphen) *Ein* **aufspannender Baum** *eines (ungerichteten, zusammenhängenden) Graphen $G = (V,W)$ ist ein zusammenhängender Teilgraph $H = (V,T)$ auf derselben Knotenmenge. Ein* **minimal aufspannender Baum** *ist dabei ein aufspannender Baum T mit minimalem Gewicht. Dabei gilt für das Gewicht eines Baumes T:*

$$w(T) := \sum_{e \in T} w(e)$$

.

Das Dekodierungsproblem lässt sich wie folgt in die Graphentheorie übertragen. Angenommen, alle Limits L_i wären berechnet, so lässt sich folgender Baum konstruieren.

Transformationsregeln:

- Die Knotenmenge E sei die Klassenmenge K
- Verbinde jedes Paar von Knoten $v, w \in E$ durch eine ungerichtete Kante (v,w)
- gewichte jede Kante $e_{v,w} = (v,w) \in E$ mit dem Wert $w(e_{v,w}) = L(k_v) + L(k_w)$.

Daraus resultiert ein ungerichteter, zusammenhängender und gewichteter Graph G_T.

Lemma 3.2.9 *Für den minimal aufspannenden Baum T des nach obigen Transformationsregeln gebildeten Graphen G_T gilt: (Unter Annahme der Eindeutigkeit des Top Rank)*
Die korrekte top rank *Klasse ist inzident zu allen Kanten von T. Darüber hinaus existiert für alle restlichen Klassen jeweils genau eine inzidente Kante.*

Beweis 3.2.6 *Es gilt:* $\forall i \in K\backslash\{top\} \rightarrow L_{top\ rank} < L_i$. *Somit gilt auch* $\forall i, j \in K\backslash\{top\} \rightarrow L_{top\ rank} + L_j < L_i + L_j$. *Das heißt, von den verbundenen Kanten für eine beliebige Klasse* $k_i \in K\backslash\{top\}$ *in* K_T *ist die zu* k_{top} *inzidente Kante minimal. Da alle inzidenten Kanten von* k_{top} *bereits einen Spannbaum von* G_T *bilden, ist keine weitere Kante notwendig.*

Lemma 3.2.10 *Die Kanten des minimal aufspannenden Baumes von* G_T *entsprechen genau der minimalen Anzahl an Klassifikatoren, die zur Verifikation des* L_{top} *und somit zur Bestimmung der* top rank *Klasse benötigt werden (siehe 3.2.4).*

Indem Klassifikatoren und Klassen auf einen Graphen abgebildet wurden, erscheint das Dekodierungsproblem mit obigem Lemma durch einen effizienten Algorithmus zur Bestimmung des minimalen Spannbaums möglich. Dafür soll der Algorithmus von Kruskal hinzugezogen werden:

Algorithmus 3 : Algorithmus von Kruskal

Input : Ein (ungerichteter, zusammenhängender, gewichteter) Graph $G_T = (G, E)$ mit den Gewichten w.

Output : ein minimaler Spannbaum von G_T

1 **begin**

2 Sortiere die Kanten in E nach aufsteigendem Gewicht: Sei $E = \{e_1, \cdots, e_m\}$ mit $w(e_1) \leq \cdots \leq w(e_m)$.

3 Setze $T := 0$, $k := 1$.

4 Falls die Kanten in $T \bigcup \{e_k\}$ einen Baum bilden, setze $T := T \bigcup \{e_k\}$.

5 Falls $k = m$, dann STOP. Sonst setze $k := k + 1$ und weiter mit 4.

6 **end**

Die Arbeitsweise des Algorithmus von Kruskal lässt sich kurzgefasst wie folgt beschreiben:

> Beginnend mit einem leeren Graphen G_E, werden iterativ, die Kanten mit dem minimalen Gewicht von G_T hinzugefügt, solange G_E azyklisch bleibt.

In den Abbildungen 3.8 und 3.9 ist grafisch die Anwendung anhand einer konkreten Instanz aus dem Datensatz `glass`, ausgehend von dem erzeugten Graphen, mithilfe der Transformationsregeln, zu betrachten. Die vollständige Auswertung aller Klassifikatoren ergab folgende Limit-Werte:

$$L_0 = 0.369, L_1 = 1.552, L_2 = 1.162, L_3 = 6, L_4 = 3.5, L_5 = 4.056, L_6 = 3.363$$

Abbildung 3.9 stellt den minimal aufspannenden Teilbaum dar und die Werte in den Klammern stellen die Reihenfolge der ausgewählten Kanten nach dem Algorithmus von Kruskal dar[4]. Jede beliebige zusätzliche Kante würde einen Zyklus im Graphen bilden.

Aus den Limit-Werten wird die Klasse 0 als top rank identifiziert. Die Aussage aus Lemma 3.2.4 kann in der Grafik beobachtet werden. Der minimale Spannbaum besitzt ein „sternenförmiges“ Muster mit der korrekten Klasse 0 als Zentrum.

[4] Kanten, die während der Iteration zu einem Zyklus führten, wie z. B. die Kante zwischen 1 und 2, wurden ausgelassen.

Abbildung 3.8: Die Abbildung der Dekodierung auf einen Graphen anhand einer konkreten Dateninstanz.

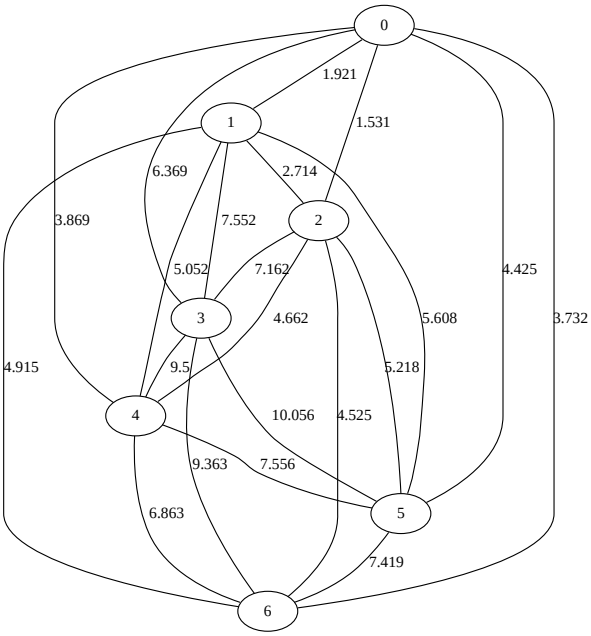
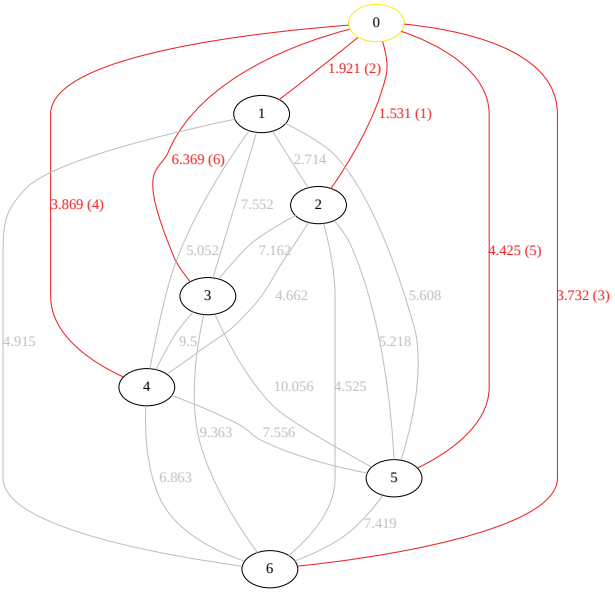


Abbildung 3.9: Lösung des Dekodierungsproblems als minimaler Spannbaum



Der Algorithmus von Kruskal ließ sich hier jedoch nur anwenden, da die korrekten Limits vorgegeben waren. In der gegebenen Problemstellung würden sich die Limits erst mit jeder Klassifikationsauswertung iterativ annähern.

Der QWeighted Algorithmus soll nun in der Graphenansicht angewendet werden. Zunächst wird dazu ein Graph aus den Transformationsregeln konstruiert. Dies bedeutet, dass zu Beginn des Algorithmus alle Kantengewichte null sind, da alle Limits null sind. Dabei sollen nun die Schritte des Algorithmus anhand des konkreten Beispiels vollzogen werden.

- **1. Schritt**
  Alle Limits sind null $L = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$, QWeighted wählt zwei beliebige Spieler k_i, k_j aus und wertet den Klassifikator $K_{i,j}$ aus, auf den Graphen übertragen wird die Kante (i, j) aus G_T zu G_S hinzugefügt. Es wird $K_{3,4}$ ausgewählt. Klasse 4 gewinnt mit einem perfekten Voting von 1. $L = [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]$.

- **2. Schritt**
  Die Auswahl der Kanten/Klassifizierer ist immer noch beliebig. Es wird $K_{4,6}$ ausgewählt, dabei gewinnt Klasse 6 mit einer Wertung von 1.0. $L = [0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0]$

- **3. Schritt**
  Der Algorithmus wählt zufällig $K_{2,6}$. Klasse 2 gewinnt mit 0.969. $L = [0.0, 0.0, 0.031, 1.0, 1.0, 0.0, 0.969]$

- **4. Schritt**
  Zur Auswahl stehen $K_{0,1}, K_{0,5}, K_{1,5}$. Es wird $K_{1,5}$ ausgewertet mit folgendem Ergebnis: Klasse 1 gewinnt mit einer Wertung von 1.
  $L = [0.0, 0.0, 0.031, 1.0, 1.0, 1.0, 0.969]$.

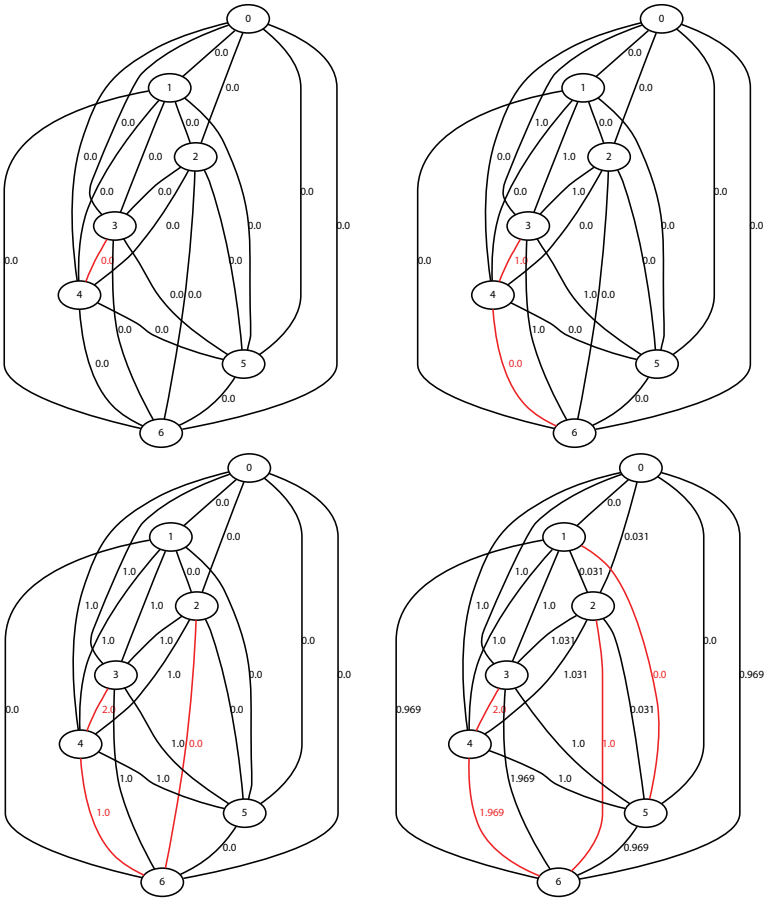
- **5. Schritt**
  Nun ist die Auswahl zum ersten Mal eindeutig. Klasse 0 spielt gegen Klasse 1 und Klasse 0 gewinnt mit 0.777.
  $L = [0.223, 0.777, 0.031, 1.0, 1.0, 1.0, 0.969]$

- **6. Schritt**
  Klasse 0 besitzt das aktuell beste Limit und wird mit Klasse 2 gepaart. Ergebnis: 0 gewinnt mit 0.8625.
  $L = [0.36, 0.777, 0.894, 1.0, 1.0, 1.0, 0.969]$

- **7. Schritt**
  Die Auswahl fällt eindeutig auf $K_{0,6}$.Klasse 0 gewinnt mit 0.991.
  $L = [0.369, 0.777, 0.894, 1.0, 1.0, 1.0, 1.96]$

- **8-10. Schritt**
  Es werden die Klassifikatoren $K_{0,4}, K_{0,5}, K_{0,3}$ in dieser Reihenfolge ausgewertet, wobei die Klasse 0 immer mit einer Wertung von 1 gewonnen hat. Die Limits:
  $L = [0.369, 0.777, 0.894, 1.0, 2.0, 1.0, 1.96]$
  $\rightarrow L = [0.369, 0.777, 0.894, 1.0, 2.0, 2.0, 1.96]$
  $\rightarrow L = [0.369, 0.777, 0.894, 2.0, 2.0, 2.0, 1.96]$.
  Für die Klasse 0 wurden alle Klassifikatoren ausgewertet. Da ihr Limit das aktuell beste darstellt und die Limits der restlichen Klassen durch weitere Klassifikationsauswertungen laut Lemma 3.2.2 sich nur verschlechtern können, kann abgebrochen werden, mit dem Ergebnis: Klasse 0 ist top rank Klasse.

Abbildung 3.10: QWeighted Algorithmus im Graphen. Schritte 1 bis 4



Wird in einem Iterationsschritt ein Klassifikator ausgewählt, bedeutet das auf den Graphen übertragen, dass die jeweilige Kante markiert wird. Die Kantengewichte des Graphen G_T ändern sich nach jeder Auswertung. Dieser Ablauf wird beginnend mit Abbildung 3.10 grafisch dargestellt.

Wie man feststellt, erzeugt der QWeighted Algorithmus einen Teilgraphen, die eine Obermenge des minimalen Spannbaums ist. Dass dies immer gilt, ist klar durch Lemma 3.2.4. Auf Graphen übertragen, kann man den QWeighted Algorithmus folgendermaßen beschreiben:

> Beginne mit einem leeren Graphen G_Q. Füge die minimale Kante aus G_T in G_Q hinzu. (Die Kantengewichte von G_T werden daraufhin aktualisiert). Wiederhole diesen Schritt, bis ein Knoten existiert, der $n-1$ inzidente Kanten besitzt. Dieser Knoten stellt dann die top rank Klasse dar.

Dieser Algorithmus lässt sich auch etwas modifizieren, falls explizit der minimale Spannbaum gewünscht wird. Dafür wären zwei Möglichkeiten denkbar. Eine Variante wäre es, nach der Anwendung des QWeighted Algorithmus einfach die nicht zur top rank Klasse inzidenten Kanten zu entfernen. Die zweite Möglichkeit eignet sich für Situationen, in denen garantiert sein muss, dass in jedem Schritt der Iteration der Graph azyklisch ist.

> Beginne mit einem leeren Graphen G_Q. Füge die minimale Kante aus G_T in G_Q hinzu. (Die Kantengewichte von G_T werden daraufhin aktualisiert). Wiederhole diesen Schritt, bis ein Knoten existiert, der $n-1$ inzidente Kanten besitzt. Falls die aktuelle Kante einen Kreis im Graphen G_Q erzeugt, entferne die Kante mit dem größten Gewicht.

Abbildung 3.11: QWeighted Algorithmus im Graphen. Schritte 5 bis 8

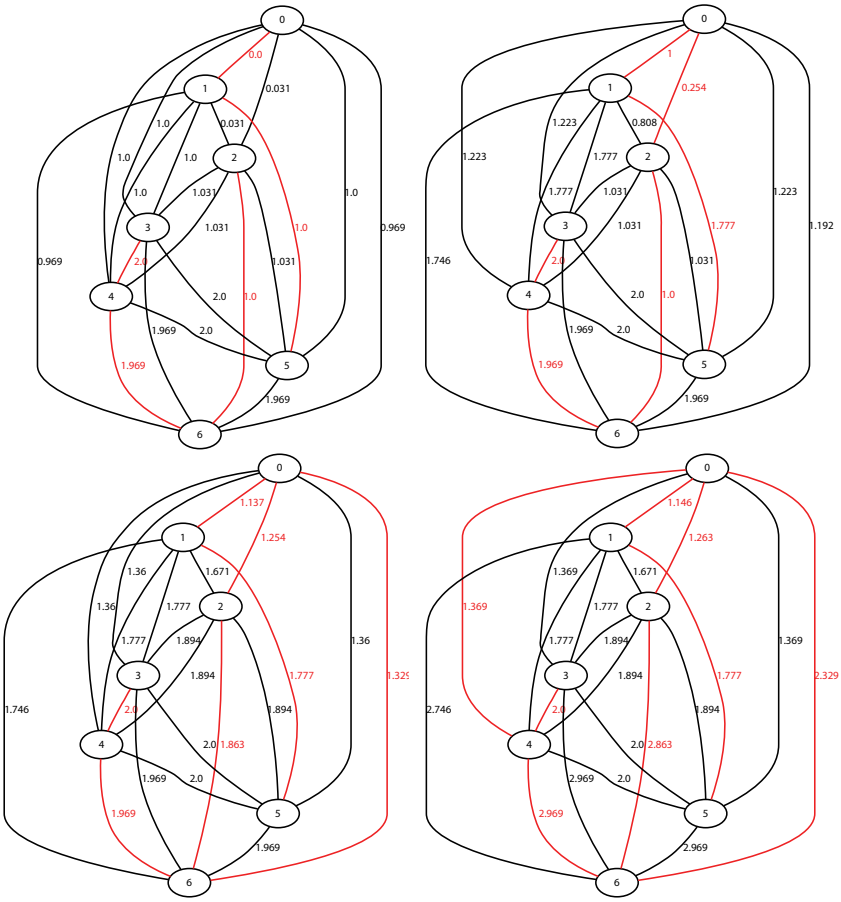
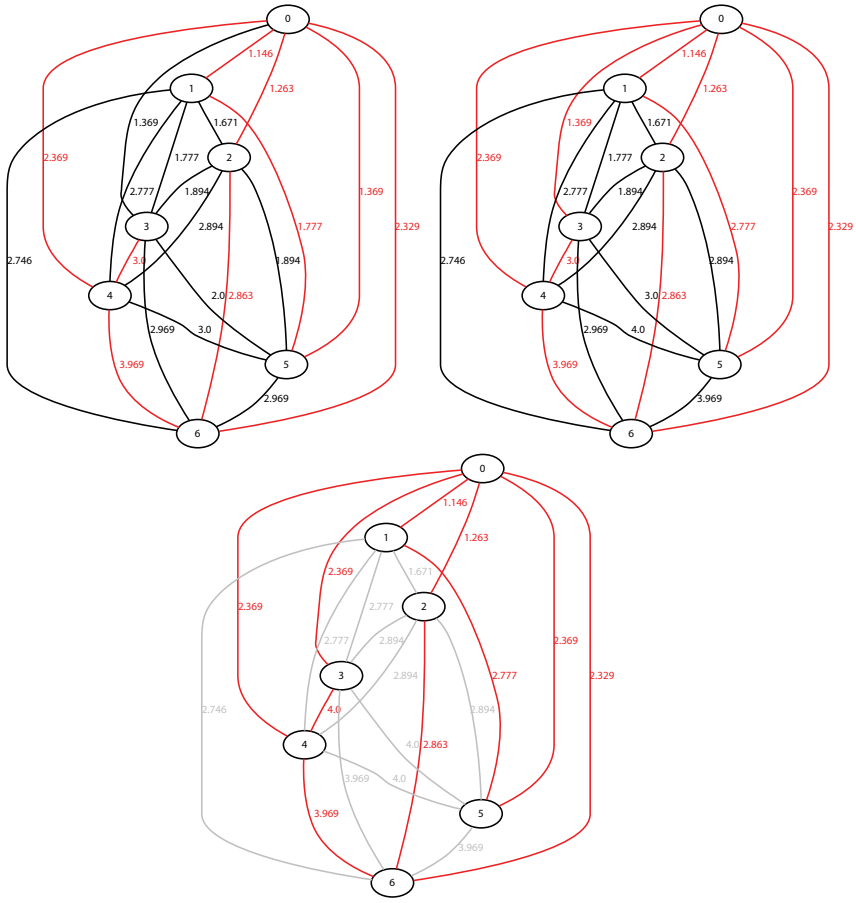


Abbildung 3.12: QWeighted Algorithmus im Graphen. Schritte 9 bis 11



Diese Neuevaluierung wird nötig, da die Kantengewichte nicht den endgültigen Werten entsprechen. Somit wird aber garantiert, dass der entstehende Graph G_Q in jedem Iterationsschritt immer azyklisch und am Ende des Algorithmus minimaler Spannbaum von G_T ist.

Aufgrund dieser Überlegungen lässt sich sagen, dass der QWeighted Algorithmus ein auf dem Greedy-Prinzip basierender Algorithmus ist, die eine spezielle Version des Kruskal Algorithmus darstellt. Sie erzeugt auf effiziente Weise eine möglichst kleine Obermenge des minimalen Spannbaumes eines Graphen, dessen Kantengewichte nicht von vornherein gegeben sind, sondern iterativ angenähert werden. Die Kantengewichte sind dabei monotone Funktionen. Es ist zusätzlich durch eine kleine Modifikation möglich, den exakten minimalen Spannbaum von solchen Graphen zu erzeugen, wobei hier nicht genauer auf ihre Komplexität eingegangen wird.

3.2.8 Zusammenfassung

Zusammenfassend besitzt *QWeighted* folgende Eigenschaften:

- Der Algorithmus ist einfach und basiert im Wesentlichen aus der Einbeziehung von kombinatorischen Aspekten einer Stimmenverteilung mit dem Detail, dass für das Klassifikationsergebnis nur der *top rank* von Bedeutung ist.

  Anders als bei *DDAG* und damit dem zugrunde liegenden *Voting-Against* Ansatz ist keine Annahme über die *falsch negativen* Fehler der Klassifikatoren notwendig.

- Das Ergebnis des *QWeighted* ist immer identisch mit dem Ergebnis einer vollständigen *Weighted-Voting* Dekodierung. Im Falle von ungewichteten Klassifikatoren (z. B. $w_{i,j} \in \{0,1\}$) ist das Ergebnis identisch einer *Voting* Dekodierung. Somit übernimmt es die gute Genauigkeit dieser Dekodierungsarten. Das heißt, aufgrund der Gleichheit sind keine erneuten Benchmarks mit anderen Dekodierungsarten notwendig, um einen Vergleich zu ziehen.

- Die Reduzierung der Anzahl der Vergleiche ist nahezu unabhängig von dem Basislerner. Die verwendete *Art* des Lernens hat keinen Einfluss auf die Effizienz des Algorithmus. Allein die Effektivität oder Genauigkeit der Basislerner wirkt sich auf die Effizienz aus.

- Im *Worst-Case* Fall ist die Laufzeit identisch mit der üblichen *(Weighted) Voting* Dekodierung ($\frac{n(n-1)}{2}$). Theoretisch ist eine *Best-Case* Laufzeit von $n-1$ möglich. Für den durchschnittlichen Fall kann man für die Praxis von einer $n \cdot \log(n)$ Laufzeit ausgehen.

- QWeighted ist ein Greedy-basierter Algorithmus, dass eine spezielle Version des Kruskal Algorithmus darstellt. Sie kann minimale Spannbäume für zusammenhängende, ungerichtete, gewichtete Graphen erstellen, wobei die Kantengewichte monotone Funktionen sind und iterativ angenähert werden müssen.

4 Effizientes Ranking

Neben der Klassifikation stellt die Generierung von Rankings eine wichtige Anwendung dar. Nicht nur die beste Klasse wird gesucht, sondern eine geordnete Liste der Klassen nach „Relevanz“. Dabei kann man die Rankinggenerierung als Verallgemeinerung der Klassifikation ansehen. Den Nutzen einer automatischen Rankinggenerierung wird deutlich, wenn man sich die vielfältigen Anwendungsgebiete bewusst macht.

Prominentestes Beispiel sind wohl Suchmaschinen. Auf eine Suchanfrage mit bestimmten Schlüsselwörtern wird nicht jede übereinstimmende Seite in einer zufälligen Reihenfolge aufgelistet, sondern mit „intelligenten“ Algorithmen in einer bestimmten Reihenfolge präsentiert. Dadurch reduziert sich natürlich der Zeitaufwand, um die gefundenen Einträge durchzugehen, bis man die passende Seite gefunden hat. Daneben gibt es zahlreiche Anwendungsfelder aus der Wirtschaft. Ein Produktmanager, der eine werbewirksame Einführung eines neuen Produktes plant, ist z. B. interessiert an einem Ranking der Werbemedien, geordnet nach Erreichbarkeit der gewünschten Zielgruppe. Dabei ist er interessiert, sein Produkt auf möglichst vielen Werbeplattformen zu platzieren. Dann wäre es nicht verkehrt, bei der Verteilung des Budgets sich an dem Ranking zu orientieren. In der Tat gibt es verwandte Felder aus der Wirtschaft wie *Analytic Hierarchy Process* [20] (*AHP*) oder *Nutzwertanalyse* [14] (*NWA*), die sich mit der Generierung von Rankings, insbesondere auch mit paarweisen Vergleichen beschäftigen.

4.1 Problemstellung

Zunächst soll der Sachverhalt formalisiert werden. Die folgende Darstellung des Problems wurde aus [9] übernommen, das als *constraint classification* vorgestellt wurde.

Gegeben:

- eine Menge der Labels $K = \{\lambda_i |\ i = 1 \ldots m\}$
- eine Menge der Testinstanzen $T = \{t_k |\ k = 1 \ldots n\}$
- für jede Testinstanz t_k:

– eine Menge von Präferenzen $P_k \subseteq K \times K$, wobei $(\lambda_i, \lambda_j) \in P_k$ impliziert, dass Label λ_i dem Label λ_j für die Testinstanz t_k vorgezogen wird (abgekürzt mit $\lambda_i \succ_k \lambda_j$).

Gesucht: eine Funktion, die eine totale Ordnung (Ranking) der Labels $\lambda_i, i = 1 \ldots m$ für jede Testinstanz liefert

Diese Darstellung stellt eine Verallgemeinerung von mehreren Machine-Learning Problemstellungen dar:

- *Ranking*: jede Trainingsinstanz ist mit einer totalen Ordnung der Labels verknüpft. Das heißt, für beliebige Paare von Labels (λ_i,λ_j) gilt entweder $\lambda_i \succ \lambda_j$ oder $\lambda_j \succ \lambda_i$.
- *Klassifikation*: zu jeder Instanz wird genau ein Label λ_i zugeordnet (entspricht den Klassen). Dadurch gilt für die Menge der Präferenzen: $\{\lambda_i \succ \lambda_j | j \leq 1 \neq j \leq c\}$.
- *Multi-Label Klassifikation*: jede Instanz e_k ist mit einer Menge von Labels $S_k \subseteq L$ assoziiert. Dann gilt: $\{\lambda_i \succ \lambda_j | \lambda_i \in S, \lambda_j \in L \backslash S\}$.

4.1.1 Transitivität der Präferenzen

Die oben eingeführten Präferenzen (λ_i, λ_j) stellen die Ergebnisse der paarweisen Klassifikationen dar. $K_{i,j} \in [0,1]$ sei der gelernte Klassifikator für die Klassen λ_i, λ_j. Für die Erstellung der Präferenzen ist eine nahe liegende Methode:

$$K_{i,j}(t_k) > 0.5 \rightarrow (\lambda_i, \lambda_j) \text{ bzw. } K_{i,j}(t_k) \leq 0.5 \rightarrow (\lambda_j, \lambda_i)$$

Werden diskrete Klassifikatoren $K_{i,j} \in \{0, 0.5, 1\} \wedge K_{i,j} + K_{j,i} = 1$ eingesetzt, wobei 0.5 einen Gleichstand bedeutet, so gelten die Beziehungen:

$$K_{i,j}(t_k) = 1 \rightarrow (\lambda_i, \lambda_j) \text{ bzw. } K_{i,j}(t_k) = 0 \rightarrow (\lambda_j, \lambda_i)$$

Für einen Gleichstand werden üblicherweise die Gleichstandsbehandlungen 3.2 und 3.3 zur Bestimmung einer Präferenz herangezogen. Diese können im Übrigen auch für einen „weighted“ Klassifikator angewendet werden, wenn man in obigen Aussagen, die $\leq$ Bedingung zu $<$ verschärft.

Da Lernverfahren nicht perfekt sind, können die Klassifikatoren Fehler aufweisen. Dies hat zur Folge, dass die Vergleiche bzw. Präferenzen nicht transitiv sind. Für den Fall, dass die Vergleiche transitiv wären, würde sich das Ranking-Problem auf ein Sortierproblem reduzieren und man könnte als Dekodierungsmethode ein von den vielen Sortieralgorithmen wie z. B. *Quicksort* benutzen.

Jedoch kann man davon ausgehen, dass die Präferenzen anti-symmetrisch und irreflexiv sind.

4.2 Überblick der Dekodierungsarten

Nachfolgend werden kurz informal die gängigen Dekodierungsarten vorgestellt, die hauptsächlich aus [10] übernommen wurden:

- Votingbasiert

Die votingbasierten Ansätze wie *Weighted Voting* liefern eine Verteilung der Stimmen über die Klassen. Die Klassen lassen sich somit über die Stimmenanzahl vergleichen und sich bezüglich der normalen "<„-Relation in eine totale Ordnung fassen. In [10] wurde anhand von synthetischen Daten die simple *Voting*-Dekodierung mit anderen fortgeschritteneren Verfahren, wie Slater-optimalen Rankings und Iterative Choice verglichen. *Voting* konnte dabei mit den laufzeitintensiveren Verfahren mithalten. Im Falle von sehr genauen Basislernern könnten die komplexeren Verfahren jedoch zu weiteren Verbesserungen der Ergebnisse führen.

- Slater-Optimal

  Slater-Optimal ist ein Begriff aus der Graphentheorie. Man geht von der Problemstellung aus, einen zyklischen Graphen in einen azyklischen Graphen umzuwandeln. Einziges Mittel ist hierbei die Invertierung der gerichteten Kanten. Die sogenannte Slater-Zahl [13] ist dabei die minimale Anzahl der Invertierungen, die benötigt werden.

  Dieser Begriff aus der Graphentheorie stellt hier einen Bezug dar, da man das Dekodierungsproblem in dieses graphentheoretische Problem umformulieren kann. Interpretiert man die gerichteten Kanten eines Graphen von beliebigen Knoten a nach b als Präferenz (λ_a,λ_b) und die Invertierung einer Kante als Invertierung einer Präferenz $(\lambda_i,\lambda_j) \rightarrow_{Invertierung} (\lambda_j,\lambda_i)$ so würden die Slater-optimalen Umwandlungen dazu führen, dass die Präferenzen transitiv sind.

  Es ist zu erwähnen, dass die Bestimmung eines Slater-Optimalen Rankings ein NP-vollständiges Problem darstellt [22].

- Sortieralgorithmen

  Für den Fall, dass die Präferenzen transitiv sind, reduziert sich die Dekodierung auf ein Sortierproblem. Somit wäre auch jede Art von Sortieralgorithmus wie z. B. *Quick Sort* eine mögliche Dekodierungsart.

- Iterative Choice

  Neben den explizit auf Ranking ausgelegten Verfahren lässt sich mit der einfachen Idee des *iterative choice* grundsätzlich jede **Klassifikations**-Dekodierung zur Rankingbestimmung benutzen. Dabei wird mit der Klassifikationsdekodierung der top rank ermittelt, diese Klasse danach aus der Klassenmenge entfernt und auf dieser Klassenmenge wieder die Klassifikationsdekodierung angewendet. Dies wiederholt sich solange, bis die Klassenmenge leer ist.

4.3 Maßstab

Bevor die Ergebnisse präsentiert werden, ist es nötig einen Maßstab für Rankings zu bestimmen. Ein Maßstab ist der sogenannte *Ranking-Error*. Er behandelt den offensichtlichsten Fall, in dem das komplette Ranking relevant ist, das heißt alle zugewiesenen Positionen der Klassen im Ranking. Ein Beispiel wäre es, einen optimalen Fahrplan für einen Lieferanten je nach bestimmten Kriterien, wie Gesamtlänge der Strecke oder Gesamtfahrzeit, vorherzusagen. Dabei ginge es also darum, eine Liste der Geschäfte zu lernen, die in einer bestimmten Reihenfolge angefahren werden. Ein anderes Beispiel wäre das Problem, die Landungen von Flugzeugen am Flughafen so zu verteilen, dass keine

unnötigen Leerlaufzeiten entstehen. Es soll dabei angenommen werden, dass gelernte Verfahren benutzt werden, da für eine genaue optimale Berechnung nicht genügend Zeit oder Rechenleistung vorhanden sei. In solchen Fällen, in denen jede Position im Ranking relevant ist, geht man von einer Abstandsmessung vom vorhergesagten Ranking zum wirklichen Ranking aus. Diese Abweichung beider Rankings stellt den *Ranking-Error* dar.

Dazu muss zunächst eine Rankingfunktion definiert werden:

Definition 4.1 (Ranking-Funktion) *Sei $i \in K$ eine Klasse aus der Klassenmenge und R ein Ranking oder Ordnung. Die Ranking-Funktion τ_R bezüglich dem Ranking R ist wie folgt definiert:*

$$\tau_R(i) := \textit{Position der Klasse i im Ranking R}$$

Definition 4.2 (Ranking-Error) *Die Differenz D_R zwischen zwei Rankings R_1, R_2 ist:*

$$D_R(R_1, R_2) := \sum_{i=1}^{m} (\tau_{R_1}(i) - \tau_{R_2}(i))^2$$

Neben dieser intuitiven Abstandsbestimmung zweier Rankings mit einer bestimmten Norm sind weitere Methoden möglich. Aufgrund der verschiedenen Anwendungsgebiete der Rankings sind andere Maßstäbe denkbar.

Ein Beispiel dafür ist der sogenannte *Position-Error*. Anders als bei dem *Ranking-Error* ist nur die Position *einer* Klasse relevant. Als einfaches Beispiel kann man sich einen Vertreter vorstellen, der einer Hausfrau etwas verkaufen will. Aus jahrelanger Erfahrung hat der Vertreter ein gewisses Ranking an Produkten verinnerlicht, nach denen er vorgeht. Angenommen an erster Stelle wäre der Staubsauger, das jedoch auf kein Interesse der Hausfrau trifft. Er geht nun alle Produkte nach seinem Ranking durch, bis die Hausfrau Interesse zeigt und es eventuell zum Verkauf kommt. Dann wäre der Position-Error genau der Abstand dieses Produktes (*korrekte top rank*) zum Anfangswert. Als weiteres Anwendungs-Szenario kann man sich ein automatisches Diagnosesystem vorstellen. Das Diagnosesystem geht bei einer Fehlfunktion in einer bestimmten Reihenfolge alle bekannten Fehlerarten durch, bis der richtige Fall erkannt wurde und leitet daraufhin die entsprechende Fehlerbehandlung ein. In diesem Fall wird angenommen, dass die entsprechende Fehlerart und somit auch Fehlerbehandlung im System existiert. In einem solchen Szenario wäre ein adäquater Maßstab für das vorhergesagte Ranking die Anzahl der Vorgänger des korrekten Fehlerartes, also dessen Position im Fehlerkatalog.

Diese Positionsabweichung des *korrekten top ranks* im vorhergesagten Ranking ist der *Position-Error*.

Definition 4.3 (Position-Error) *Sei i die korrekte top rank Klasse. Der Position-Error von den Rankings R_1 und R_2 ist dann wie folgt definiert:*

$$D_P(R_1, R_2) := |\tau_{R_1}(i) - \tau_{R_2}(i)|$$

4.4 Problematik bei der Evaluierung von Rankings

Rankinggenerierung ist ein noch recht junges Feld in dem Zweig des Maschinellen Lernens. Ein größeres Problem bei der Entwicklung neuer Ranking-Algorithmen oder Evaluierung bestehender, ist die Unvollständigkeit der Daten. Die verfügbaren Trainings- und Test-Datensätze bestehen meistens lediglich aus einem korrekten Klassenlabel. Das heißt, in der Regel sind keine Datensätze gegeben, die ein vollständiges Ranking beinhalten, das als Referenz für Analysen dienen könnte. Vor diesem Problem standen auch die Auswertungen des „ Schweizer-Systems “ in dieser Diplomarbeit.

In dieser Diplomarbeit wurde folgendermaßen vorgegangen. Als Referenz diente das Ranking eines kompletten Weighted Voting. Dabei wurden diese Referenzrankings nicht nur für jede einzelne Instanz des Datensatzes erzeugt, sondern zusätzlich für jeden einzelnen Folding-Vorgang im verwendeten 10-fach Cross-Validation. Dies ist erforderlich, da durch Cross-Validation andere Klassifikatoren erzeugt werden (und somit Rankings), als wenn die Klassifikatoren ausschließlich auf den kompletten Trainingsdaten gebildet werden.

4.5 Das „Schweizer-System“

Der Ansatz der Diplomarbeit ist das sogenannte „Schweizer-System“. Es handelt sich dabei um eine Turnierform, die bei Schach und ähnlichen Sportarten benutzt wird. Es wurde von Julius Müller erfunden und kam erstmals 1895 zum Einsatz. Das System kommt meist bei Turnieren mit großer Teilnehmerzahl zum Einsatz, da anders als beim üblichen Round-Robin System, nicht alle möglichen Paarungen der Spieler gespielt werden. Dabei ist stark vereinfacht die Grundidee:

- In jeder Runde spielen möglichst gleich starke Gegner.
- Die Rundenzahl ist begrenzt.

Das heißt, in jeder Runde werden $\lfloor \frac{n}{2} \rfloor$ Spiele gespielt, wobei n die Anzahl der Teilnehmer sei, und die Rundenanzahl ist typischerweise begrenzt durch $\log(n)$. Daraus folgt, dass anstatt einer quadratischen (Round-Robin) eine $n \cdot \log(n)$ Anzahl von Paarungen stattfindet. Dies stellt eine beträchtliche Reduzierung der Spielanzahl dar und somit eine starke Reduzierung der Turnierlaufzeit. Es ist noch zu erwähnen, dass keine Paarung doppelt gespielt wird.

In dem offiziellen Regelbuch der internationalen Schach-Vereinigung (FIDE - Fédération Internationale des Échecs, World Chess Federation) ist die Regelung weitaus komplexer, z. B. wird darin der Vorteil für den Spieler mit Weiß[1] beachtet, daneben viele weitere spezielle Regeln. Hier wurde nur die grundlegende Idee dargestellt. Für weitere Details kann man in [26] die aktuelle offizielle Regelung nachschauen.

Wie oft erwähnt wurde, lässt sich das Dekodierungsproblem übertragen auf Turniersysteme aus dem Sport. Eine Reduzierung der Spielanzahl beim „Schweizer-System“ bedeutet eine Reduzierung der paarweisen Vergleiche bei der Dekodierung und darüber hinaus erscheint die $n \cdot \log(n)$- Komplexität viel versprechend. Aus diesem Grund wurde untersucht, inwieweit sich mithilfe des

[1] In der Schachwelt besteht allgemein Einigkeit darüber, dass der anfangende Spieler in Weiss einen Vorteil besitzt.

„Schweizer-System„ eine Reduzierung der Vergleichsanzahl auf die Genauigkeit der Rankingbestimmung auswirkt.

4.5.1 Auswertungen

Alle Auswertungen wurden in dem WEKA-Framework [25] durchgeführt. Dabei wurde die Klasse `MultiClassClassifier.java` um das Schweizer-System erweitert. Das Schweizer-System wurde nicht exakt nach den offiziellen Regeln mit allen Ausnahmen und Regeln abgebildet, sondern nur die Kernidee davon, dass sich folgendermaßen darstellt:

Algorithmus 4 : Schweizer-System

```
1 begin
2     for runde = 1 to max_runden do
3         bilde Paarungen von gleichstarken Spielern, bevorzuge Spieler mit
          Freilos
4         bilde Paarungen von bisher ungepaarten Spielern mit ungepaarten
          schwächeren/stärkeren Spielern
5         übrig gebliebene Spieler bekommen ein Freilos
6         spiele Paarungen und aktualisiere Statistiken (Punktzahl, Anzahl
          Spiele,...)
7 end
```

Ein Spieler bekommt ein Freilos, wenn keine Paarung für ihn ermittelt werden konnte. Das heißt, er setzt eine Runde aus. Dass solche Situationen mitunter vorkommen, wird später in einem Beispiel verdeutlicht. Spieler mit einem Freilos erhalten anders als in der offiziellen Regelung keine Punkte. Dafür wird aber garantiert, dass diese Spieler in der nächsten Runde eine Paarung spielen.

In dem obigen Pseudocode wurde bewusst das Wort „gleichstark“ benutzt, und nicht „punktgleich“, wie es in der offiziellen Regelung lautet. Das hat den Grund, dass in der Implementation anders als in der Regelung die Spielstärke nicht nach der absoluten Punktzahl gewertet wird, sondern nach dem Verhältnis $\frac{\text{Punktzahl}}{\text{Anzahl gespielter Paarungen}}$. Dies führte in Tests zu besseren Ergebnissen. Als Beispiel: Spieler A mit zwei Punkten (2 Siege) nach 2 Paarungen ist nach FIDE gleichstark wie Spieler B mit zwei Punkten nach 10 Paarungen. In der Implementation hingegen ist Spieler A deutlich stärker als Spieler B. Nichtsdestotrotz in Konstellationen, in denen die Spielanzahl aller Spieler hauptsächlich identisch ist, verliert diese Modifikation seine Bedeutung. Es wird zwar versucht mit Bevorzugung der Freilosspieler, die Spielanzahl aller Spieler nach jeder Runde gleichmäßig zu halten, jedoch kann man diese Situationen nicht ganz ausschließen. Dessen ungeachtet lässt die Modifikation eine etwas feinere Abstufung zu.

Neben den trivialen Fall für eine Freilos-Situation, dass die Spieleranzahl ungerade ist, hier ein Beispiel für eine gerade Anzahl an Spielern:

Beispiel 4.1 (Freilos-Situation 1) *Das Turnier bestehe aus 6 Spielern A, B, C, D, E, F. Zu Beginn des Turniers betragen die Punkte* $P_A = P_B = P_C = P_D = P_E = P_F = 0$. *In der 1.Runde werden die Paarungen* $(A,B),(C,D),(E,F)$ *ausgetragen. Das Ergebnis sei:* $P_A = 1, P_B = 0, P_C = 1, P_D = 0, P_E = 1, P_F = 0$.
Nächste Runde existieren zwei Punktgruppen. A,C,E *mit einem Punkt und* B,D,F *mit null Punkten. Laut Algorithmus 4 werden zunächst Paarungen aus gleichstarken Spielern bestimmt. Angenommen es würden* (A,C) *für Punkt-*

gruppe 1 und (B,D) für Punktgruppe null ausgewählt. Im zweiten Schritt wird versucht, die restlichen Spieler zu paaren. Da die Paarung (E,F) bereits gespielt wurde, ist keine Paarung für E und F zu finden. Dadurch erhalten beide ein Freilos.

Natürlich wäre für das konkrete obige Beispiel eine Runde ohne Freilos denkbar, indem eine andere Auswahl getroffen wurde. Es erscheint somit zunächst möglich den Algorithmus zu modifizieren, sodass kein Spieler in einer Runde ungepaart bleiben muss. Abgesehen von einer Erhöhung der Komplexität einer solchen Modifikation zeigt folgendes Beispiel, dass dies nicht immer der Fall ist:

Beispiel 4.2 (Freilos-Situation 2) *Man gehe von derselben Situation wie in Beispiel 4.1 aus. Jedoch wurden für die 2. Runde Paarungen gebildet, die ohne Freilos auskommen: $(A,C),(D,F),(E,B)$ mit dem Ergebnis $P_A = 1, P_B = 1, P_C = 2, P_D = 0, P_E = 1, P_F = 1$. Nun kommt es in der nächsten Runde zwangsläufig zu einer Freilos-Situation. Gleichgültig welche Paarungen für Punktgruppe 1 gebildet werden $(A,E),(B,F)$ oder $(A,F),(B,E)$, es bleiben die Spieler C und D übrig. Da die Paarung (C,D) bereits in der ersten Runde gespielt wurde, bekommen beide ein Freilos.*

Bestimmung von gleichstarken Spielern

Es ist darauf hinzuweisen, auf welche Art die Bestimmung von gleichstarken Spielern in der Implementation verläuft, da sie eine Approximation darstellt. Hauptsächlich basiert sie auf eine absteigende Sortierung der Spieler nach den vorläufigen Punktewertungen. Daraufhin werden beginnend mit dem Erstplatzierten ein ungespielter Spieler gesucht, der als nächstes in der Sortierung vorkommt. Die nach dem Erstplatzierten folgenden Spieler stellen genau die Spieler dar, die von der Punktewertung her dem Erstplatzierten am nächsten kommen. Wird ein Gegenspieler gefunden, wird die Paarung gebildet und beide aus der Sortierung entfernt. Dies wiederholt sich, bis die sortierte Liste leer ist, oder bis für die restlichen Spieler keine Paarungen möglich ist. Dies hat den einen Vorteil, dass die Bevorzugung von Freilosspieler, durch eine einfache temporäre Änderung der Punkte, vor der Sortierung möglich ist. Konkret bedeutet das, dem Freilosspieler wird ein hoher Wert zugewiesen, um ihn in der entstehenden Sortierung an erster Stelle zu platzieren. Diese einfache iterative Art der Implementation wurde für alle Auswertungen benutzt.

Nachteil dieser Methode ist sicherlich, dass nicht wirklich in Reihenfolge die „gleichstärksten“ Spieler als erstes gepaart werden. Die Punktedifferenz zwischen dem Erstplatzierten und Zweitplatzierten ist nur minimal für alle möglichen Paarungen des Erstplatzierten und nicht für beliebige Klassenpaare. Z. B. könnte der Zweit- und Drittplatzierte einen geringeren Abstand besitzen, somit „gleichstärker“ sein als Erst- und Zweitplatzierter. Das verwendete Verfahren wird jedoch die erste Paarung vornehmen. Zudem gilt, dass der Freilosspielers zum nachfolgenden Spieler in der Liste nicht unbedingt die minimale Punktedifferenz besitzt, da er künstlich nach vorne platziert wurde. In solchen Fällen wird die Sicherstellung der Gleichmäßigkeit von gespielten Spielen über den Punktegleichgewicht der Paarungsspieler gesetzt. Diese Art der Spielerauswahl war ein Versuch, einen Kompromiss zwischen den Ideen des Schweizer-Systems und der Effizienz herzustellen.

Auswertungen zum Ranking-Error

In den Abbildungen 4.1 und 4.2 wurden die rundenbasierten Ergebnisse für zwei Datensätze grafisch dargestellt.

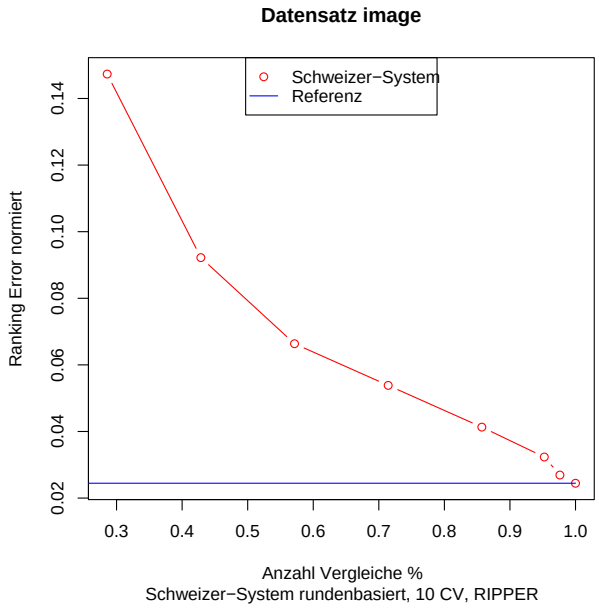


Abbildung 4.1: Ranking-Error des Schweizer-Systems auf `image`

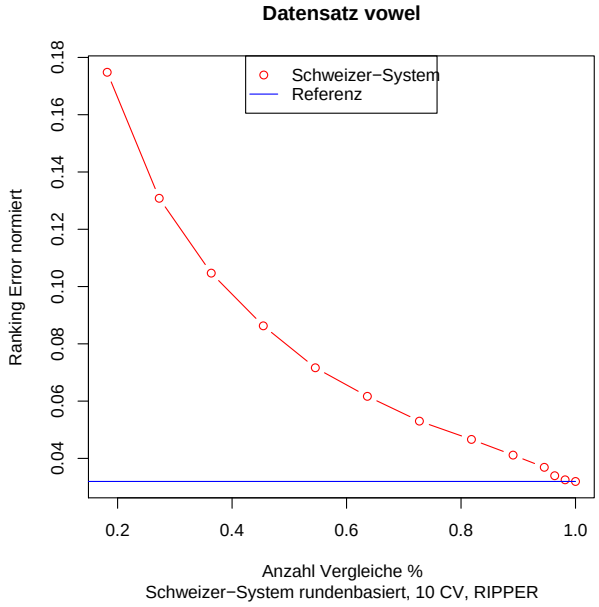


Abbildung 4.2: Ranking-Error des Schweizer-Systems auf `vowel`

Wie bereits erwähnt wurde, stellt das Rankingergebnis eines kompletten *Weighted Voting* die Referenz dar und ist in den Grafiken als blaue Linie erkennbar. Die Referenzlinie hat nicht den y-Wert von 0 (also ein Ranking-Error von 0), da durch Cross-Validation die Ergebnisse nicht perfekt sind. Die roten Kreise stellen die Ergebnisse des Schweizer-Systems nach jeder Runde dar. Dabei steht die x-Achse für die Anzahl der Vergleiche im Verhältnis zu allen möglichen paarweisen Vergleichen ($\frac{n(n-1)}{2}$). Die y-Achse gibt Auskunft über den normierten Ranking-Error, deren Bereich von 0 bis 1 verläuft. Ein Ranking-Error von 1 bedeutet ein Rankingergebnis, das komplett das Gegenteil vom korrekten Ranking ist.

Wie man feststellt, erzeugt das rundenbasierte Schweizer-System wenige Auswertungspunkte, maximal ungefähr $\frac{\frac{n(n-1)}{2}}{\lfloor \frac{n}{2} \rfloor}$, aufgrund der geringen Klassenanzahl der Datensätze. Diese Abschätzung über die Anzahl der Auswertungspunkte ist nicht exakt, da aufgrund von Freilos-Situationen pro Runde nicht immer genau $\lfloor \frac{n}{2} \rfloor$ Paarungen gespielt werden.

Bei einem 10-Klassenproblem z. B. würden bei perfekten Bedingungen (es treten keine Kollisionen/Freilossituationen auf) genau 9 Auswertungspunkte geliefert. Um eine feineren Vergleich zu erstellen, wurde das Schweizer System in einer leicht modifizierten Version ausgewertet. Es wurden nicht Auswertungen nach jeder Runde vorgenommen, sondern nach jedem Vergleich. Zusätzlich wurde dem Ranking-Error vom Schweizer-System direkt der Referenz-Wert subtrahiert. Somit bildet die x-Achse die Referenzlinie.

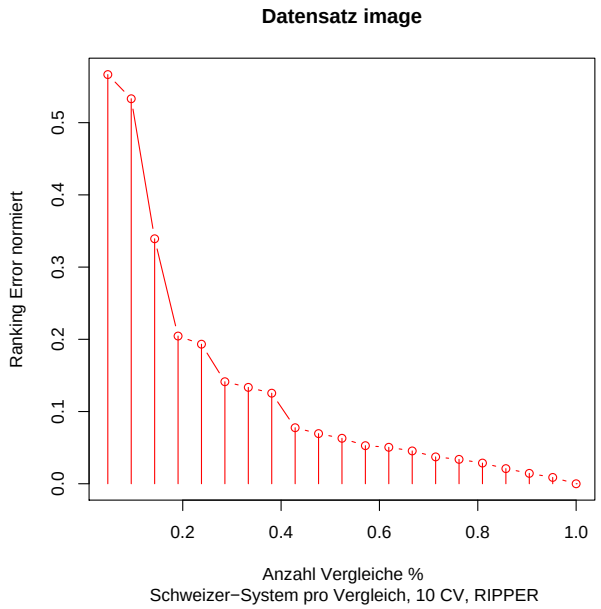


Abbildung 4.3: Ranking-Error des Schweizer-Systems auf `image` *(pro Vergleich)*

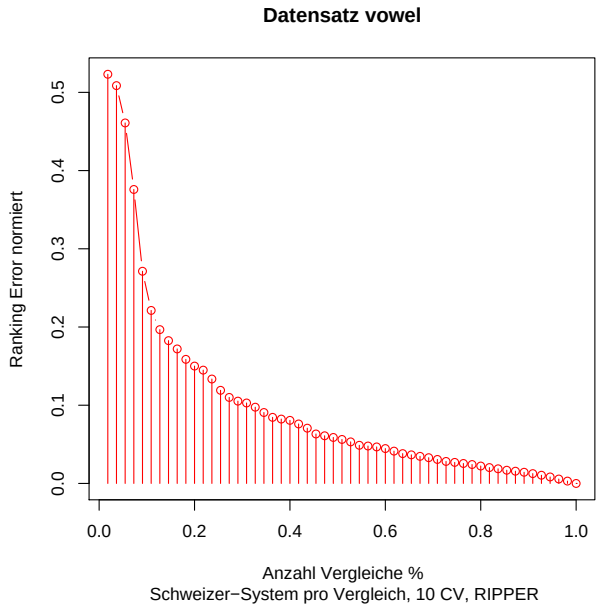


Abbildung 4.4: Ranking-Error des Schweizer-Systems auf `vowel` *(pro Vergleich)*

Des Weiteren ist zu erwähnen, dass in allen Auswertungen die Rundenzahl nicht durch $\log(n)$ beschränkt wurde, das heißt, das Schweizer-System läuft

solange, bis keine Klassifikatoren mehr übrig sind.

An allen Grafiken fällt auf, dass die Kurven konkav sind. Darüber hinaus erscheint bei geringer Anzahl von bisherigen Vergleichen die Ranking-Error Reduzierung eines zusätzlichen Vergleiches größer, als nach vielen Vergleichen. Diese quadratische Eigenschaft der Kurve lässt sich hauptsächlich auf den quadratischen Charakter der Ranking-Error Funktion zurückführen. Aus diesem Grunde wurden in den nachfolgenden Abbildungen 4.5 und 4.6 eine logarithmische Skalierung der x-Achse vorgenommen.

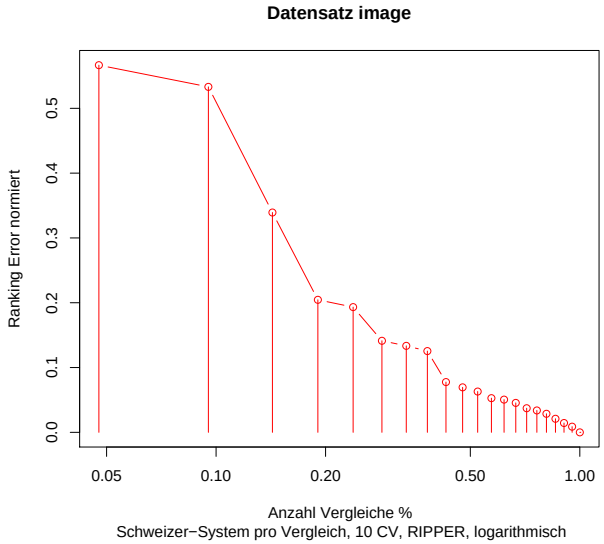


Abbildung 4.5: Ranking-Error des Schweizer-Systems auf `image` (pro Vergleich, logarithmisch)

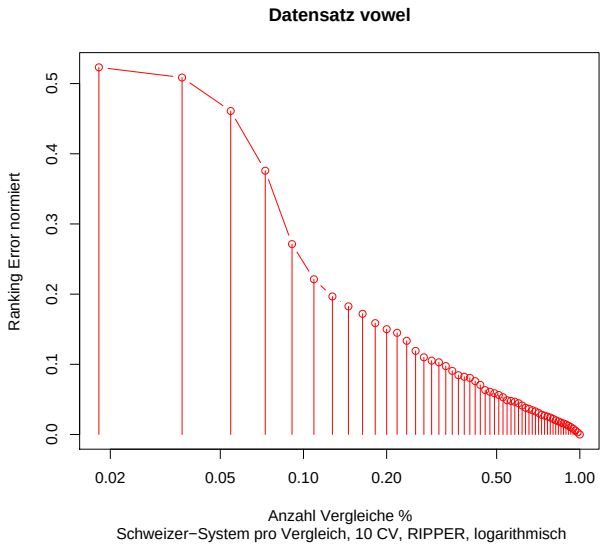


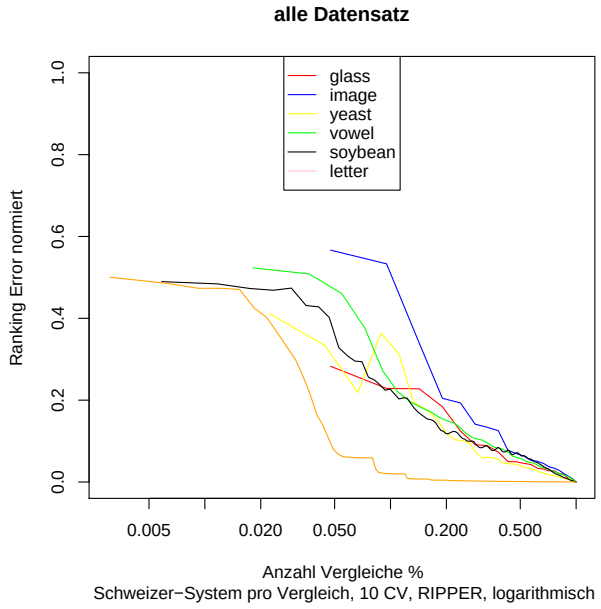
Abbildung 4.6: Ranking-Error des Schweizer-Systems auf `vowel` (pro Vergleich, logarithmisch)

Diese grafischen Modifikationen erlauben nun eine bessere Interpretation des Schweizer-Systems zu. Abgesehen von den anfänglichen Schwankungen lässt sich daraus informal interpretieren, dass *jeder Vergleich im Schweizer-System eine ungefähr gleiche Reduzierung des (normierten) Ranking-Errors verursacht.* Durch die logarithmische Skalierung der x-Achse trifft diese Aussage nicht auf den in 4.2 definierten Ranking-Error zu. Es wird hierbei vielmehr auf einen strengeren Ranking-Error in der Form $D_R(R_1,R_2) := \sum_{i=1}^{m}(\tau_{R_1}(i) - \tau_{R_2}(i))$ Bezug genommen. Die Überlegung war, dass gleich in welcher Reihenfolge die Vergleiche durchgeführt werden, ein Vergleich zweier Klassen im Durchschnitt die Position dieser beiden Klassen im Ranking verbessert wird

und somit die Ranking-Error-Kurve (mit 4.2) immer einen konvexen quadratischen Verlauf hat. Für eine effiziente Rankinggenerierung mittels Schweizer-System wäre die Verbesserung der Position mehrerer Klassen pro Vergleich im Ranking nötig, oder zumindest eine höhere durchschnittliche Verbesserung der Position beider Klassen[2].

Dass die Aussage über das Verhältnis zwischen Vergleichsreduzierung und Genauigkeitsverlust auf (fast) alle ausgewerteten Datensätze zutrifft, lässt sich in Abbildung 4.7 betrachten. Nur bei dem Datensatz `letter` ist kein linearer Verlauf zu erkennen. Dies liegt wahrscheinlich an der verwendeten Validierungsart. `letter` war der einzige Datensatz, der anhand eines Testdatensatzes überprüft wurde. Der Autor vermutet, dass dieser Testdatensatz eine signifikant „einfachere“ Klassifikation erlaubt. Dass heißt es stehen genauere Klassifikatoren zur Verfügung, die zu einem signifikant besseren Ergebnis führen, als mit 10-fach Cross-Validation. Aufgrund des immensen Zeitaufwandes eine 10-fach Cross-Validation bei `letter` auszuführen, der mit 26 Klassen und insgesamt 16.000 Testinstanzen der größte Datensatz ist, wurde in dieser Diplomarbeit darauf nicht näher eingegangen. Die Auswertungsergebnisse für die restlichen Datensätze sind im Anhang A.1 zu finden.

Abbildung 4.7: Ranking-Error des Schweizer-Systems auf alle Datensätze (pro Vergleich, logarithmisch)



Bis hierhin war jedoch nicht ersichtlich, inwiefern das Schweizer-System der normalen Round-Robin Paarungsweise überlegen ist. Um Vergleiche zu einer *zufälligen* oder in irgendeiner Art *geordneten* Reihenfolge zu ziehen, wurden anhand des Datensatzes `vowel` folgende Reihenfolgen ausgewertet:

- aufsteigende Reihenfolge. Zunächst werden alle Paarungen der ersten Klasse gespielt, danach mit der zweiten, usw.:

$$(1,2),(1,3),\cdots,(1,n),(2,3),(2,4),\cdots$$

- absteigende Reihenfolge. Alle Paarungen mit der letzten Klasse werden

[2] Es soll jedoch angemerkt sein, dass diese verwendete Betrachtung eines strengeren Ranking-Errors erst am Ende der Diplomarbeit auffiel und sich als größerer Denkfehler erwies. In der Tat wären Auswertungen mit einem Ranking-Error $D_R(R_1,R_2) := \sum_{i=1}^{m}(\tau_{R_1}(i) - \tau_{R_2}(i))$ sinnvoller gewesen.

ausgewertet, danach mit der zweitletzten, usw.:

$$(n,n-1),(n,n-2),\cdots,(n,1),(n-1,n-2),(n-1,n-3),\cdots$$

- diagonale Reihenfolge

$$(2,1),(3,1),(3,2),(4,3),(4,2),(4,1),\cdots$$

- zufällige Reihenfolge

Als Ordnungszahlen der Klassen wurden einfach die Reihenfolgen der Auflistung in den Datensätzen verwendet.

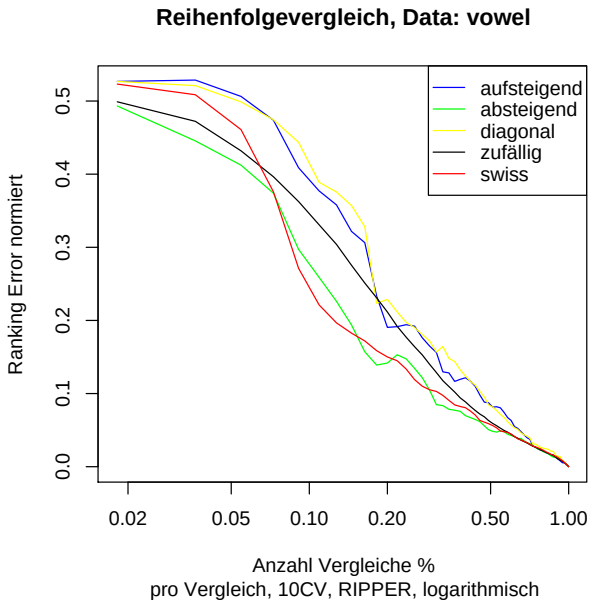


Abbildung 4.8: Ranking-Error Vergleich vom Schweizer-System und andere Reihenfolgearten auf `vowel` (pro Vergleich, logarithmisch)

Abbildung 4.8 zeigt das Ergebnis der Auswertung für den Datensatz `vowel`. Auffallend ist, dass für diesen Datensatz die absteigende Reihenfolge eine überraschend gute Reihenfolge darstellt. Jedoch kann man nicht davon ausgehen, dass dies auch für den Allgemeinfall gilt. Es ist eher anzunehmen, dass speziell in diesem Datensatz die absteigende Reihenfolge der Klassen, wie sie in dem Datensatz aufgelistet wurden, eine gewisse Charakteristik der Daten widerspiegelt. Die Auswertungen der anderen Datensätze in A.2 zeigen ähnliche Erscheinungen. Mal erscheint die absteigende Reihenfolge am geeignetsten, mal die aufsteigende. Für eine allgemeine Bewertung wird daher der Vergleich zur zufälligen Reihenfolge (schwarze Kurve) am stärksten gewichtet.

Tabelle 4.1: Ranking-Error mit verschiedenen Reihenfolgearten im Verhältnis zum Schweizer-System. Die Spalten geben den Ratio-Wert der verschiedenen Reihenfolgearten im Verhältnis zum Schweizer-System an. Dabei bedeutet ein Wert größer als 1 eine schlechtere Genauigkeit und kleiner als 1 eine bessere.

Name	aufsteigend	absteigend	diagonal	zufällig
vehicle	1.037	1.065	1.115	1.074
glass	1.033	1.120	1.252	1.154
image	0.979	1.167	1.067	1.042
yeast	0.934	1.215	1.221	1.164
vowel	1.109	0.980	1.151	1.043
soybean	1.008	1.079	1.108	1.013
letter	1.036	1.165	1.111	1.083

In Tabelle 4.1 ist ein Vergleich der Verhältnisse zwischen den verschiedenen Reihenfolgearten zu sehen. Die Werte wurden folgendermaßen gebildet.

Der Ranking-Error $D_R(R_{\text{Reihenfolgeart}}, R_{\text{Referenz}})$ einer Reihenfolgeart für jede Anzahl i an bisherigen Vergleichen wurde mit dem des Schweizer-Systems ($D_R(R_{\text{swiss}}, R_{\text{Referenz}})$) in Verhältnis gesetzt. Das Verhältnis wurde wie folgt berechnet:

$$Ratio_i := \frac{D_R(R_{\text{Reihenfolgeart}}, R_{\text{Referenz}})}{D_R(R_{\text{Reihenfolgeart}}, R_{\text{Referenz}}) + D_R(R_{\text{swiss}}, R_{\text{Referenz}})}$$

Dieser Wert wurde noch zusätzlich mit 2 multipliziert und hat damit den Wertebereich von [0,2]. Ein Wert von 1 bedeutet ein identischer Ranking-Error wie das vom Schweizer-System. Der Mittelwert der $\frac{n(n-1)}{2}$ Ratio-Werte für jeden Datensatz wurde gebildet und ist in der Tabelle zu betrachten. Grob geschätzt lässt sich sagen, dass die „geschickte“ Reihenfolge, die das Schweizer-System darstellt, besser als die anderen Reihenfolgen ist, im Hinblick auf den Allgemeinfall. *Das Schweizer-System ist oft besser als die anderen Reihenfolgearten, insbesondere ist sie immer besser als eine zufällige Reihenfolge.*

Auswertungen zum Position-Error

Da für die meisten Anwendungsgebiete wahrscheinlich der Position-Error vom korrekten *top rank* relevant ist, wurden alle Auswertungen mit dieser Problemstellung durchgeführt.

Anders als beim Ranking-Error ist für den Position-Error keine logarithmische Skalierung nötig. Es wurden wieder beispielhaft die Ergebnisse für die Datensätze `image` und `vowel` dargestellt (Abbildung 4.9 und 4.10). Eine Auflistung der Werte und Ergebnisse der restlichen Datensätze ist im Anhang A.3 zu finden.

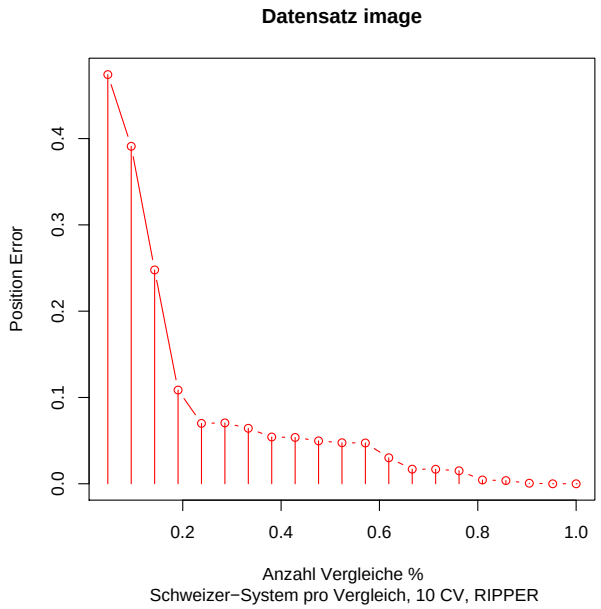


Abbildung 4.9: Position-Error vom Schweizer-System auf `image` (pro Vergleich)

Man kann erkennen, dass der Position-Error sehr stark gegen 0 konvergiert, am Ende hin sogar identisch mit 0 ist. Das heißt, für einen perfekten Position-Error sind nicht alle Vergleiche notwendig.

Betrachtet man alle Position-Error Verläufe der Datensätze in Abbildung 4.11 wird eine Beziehung zwischen Klassenanzahl und benötigter Anzahl an Vergleichen für einen guten Position-Error deutlich. Die Auflistung der Kurvenfarben in der Legende der Abbildung ist dabei in aufsteigender Klassenanzahl

Abbildung 4.10: Position-Error vom Schweizer-System auf `vowel` *(pro Vergleich)*

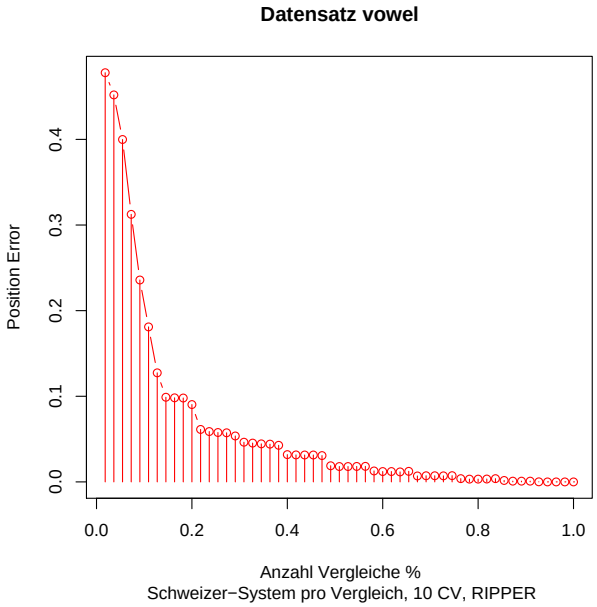
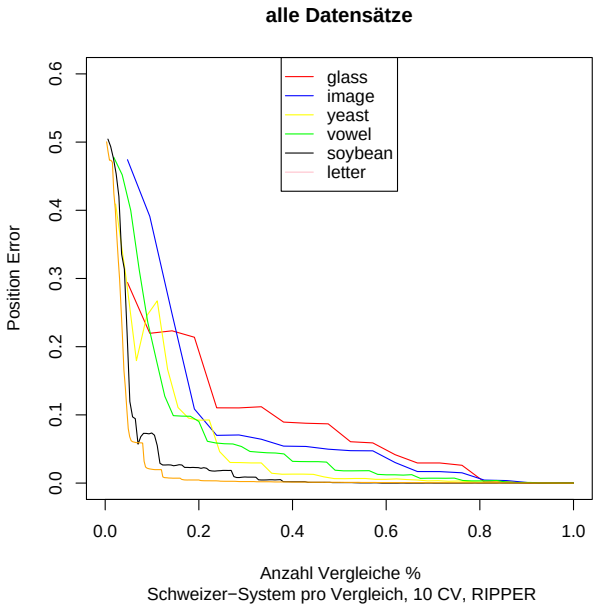


Abbildung 4.11: Position-Error vom Schweizer-System auf alle Datensätze (pro Vergleich)



geordnet, sodass man leicht erkennen kann, dass für ein Datensatz mit einer geringeren Klassenanzahl wie z. B. `image` (7 Klassen, blaue Kurve) prozentual mehr Vergleiche benötigt, um den gleichen Position-Error zu erreichen, wie `vowel` (11 Klassen, grüne Kurve). Das heißt, dass in der Reihenfolge, wie die Farben in der Legende definiert sind, die entsprechenden Position-Error Kurven stärker gegen den Nullwert konvergieren. Aus dieser Beobachtung folgt die Aussage, dass *die Genauigkeit des Position-Errors in einem umgekehrt proportionalen Verhältnis zwischen Anzahl der benötigten Vergleiche und Klassenanzahl* steht. Da hier der Position-Error vom *top rank* betrachtet wurde, bedeutet das nichts anderes, als dass für die Klassifikation nicht alle Vergleiche notwendig sind (siehe Kapitel 3.2 über QWeighted).

Tabelle 4.2: Position-Error vom Schweizer-System im Verhältnis zur Vergleichszahl und Klassenanzahl. Die erste Spalte beschreibt die Klassenanzahl. Die zweite und dritte Spalte steht für die durchschnittlich benötigte Vergleichsanzahl um einen Position-Error von $D_P < 0.02$ bzw. $D_P = 0$ zu erhalten.

Name	n	$D_P < 0.02$	$D_P = 0$
vehicle	4	5 (83%)	6 (100%)
glass	7	17 (81%)	19 (90%)
image	7	14 (67%)	20 (95%)
yeast	10	16 (36%)	44 (98%)
vowel	11	27 (49%)	52 (95%)
soybean	19	39 (23%)	145 (85%)
letter	26	34 (10%)	315 (97%)

Tabelle 4.2 verdeutlicht diese Aussage. Für einen durchschnittlichen Position-Error kleiner als 0.02, welches eine gute Genauigkeit darstellt, werden, je größer die Klassenanzahl ist, prozentual gesehen weniger Vergleiche benötigt[3]. Dieses Verhältnis trifft jedoch nicht mehr auf einen perfekten Position-Error zu. Es ist wahrscheinlich, dass durch weitere Optimierung des Schweizer-Systems bezüglich Klassifikation mit ähnlichen Ideen aus dem *QWeighted*-Algorithmus, sich auch die benötigte Vergleichszahl für einen *perfekten* Position-Error weiter reduzieren lässt.

In der Tat waren diese Beobachtungen auch die Motivation für den QWeighted Algorithmus. Anders als beim *DDAG*-Algorithmus, bei dem man mit nur $n-1$ Vergleichen eine relativ gute Klassifikation (siehe [18]) erreichen konnte, besagt diese Beobachtung, dass man dasselbe gute Ergebnis eines *kompletten Weighted Voting* auch mit weniger Vergleichen erreichen kann.

Fazit

Man konnte aus den vorangehenden Ergebnissen sehen, dass das Schweizer-System, die Ranking-Error Kurve schneller gegen den Referenzwert konvergieren lässt, als eine zufällige oder (Klassen-)geordnete Reihenfolge. Es erscheint jedoch nicht einfach eine klare Aussage zu geben, ob das Schweizer-System eine Reduzierung der Vergleichszahl mit akzeptablem Genauigkeitsverlust ermöglicht. Das liegt vor allem daran, dass unklar ist, welches Mass eine akzeptable Genauigkeit darstellt. Steht dieses Kriterium nur unter dem Einfluss eines prozentualen Genauigkeitsverlustes oder dem Verhältnis zwischen Genauigkeitsverlust relativ zur Vergleichsanzahl? In diesem Zusammenhang erscheint die zweite Beobachtung aus den Auswertungen interessant. Demzufolge ist von einem linearen Verhältnis von Vergleichsreduzierung und Genauigkeitsverlust auszugehen. Zur Verdeutlichung, eine Vergleichsreduzierung um einen Vergleich führt zu dem Verlust der Genauigkeit um die Einheit, die

[3] Eine Ausnahme bildet hier der Datensatz `vowel`. Die prozentualen Verhältnisse variieren selbstverständlich, je nach Wahl des betrachteten Schwellwertes (hier: 0.02). Jedoch war die Tendenz bereits in der Abbildung 4.11 beobachtbar.

einen Vergleich ausmacht. Das Entscheidende ist jedoch, dass diese Genauigkeitseinheit konstant ist. Somit neigt der Autor zur Verneinung der Hypothese, unter der Annahme, dass die Priorität der Vergleichsreduzierung mit der eines Genauigkeitsverlustes gleichgestellt sei. Eine logarithmisch abfallende Kurve würde indessen zu einer Fortführung dieser Fragestellung führen.

4.6 Erweitertes Schweizer-System

Im folgenden Kapitel werden Erweiterungen für das Schweizer-System vorgestellt, die zur Effizienzsteigerung dienen soll. Die Informationen zu der Buchholz und Sonneborn-Berger Feinwertung im Kapitel 4.6.2 stammen hauptsächlich aus [6].

4.6.1 Vererbung

Das Schweizer-System liefert eine gute Reihenfolge von Klassifikationsauswertungen, die durch die Wertungen der einzelnen Spieler beeinflusst wird. Die folgende Modifikation war der Versuch, die Bewertung dahingehend zu modifizieren, dass eine schnellere Konvergenz erzielt werden könnte.

Hauptidee für diese Erweiterung ist die Annahme, dass die Ergebnisse der paarweisen Vergleiche eher transitiv bezüglich des Rankings sind.

$$P(k_i >_\tau k_j | k_i \succ k_j) > 0.5$$

Ein einfaches Beispiel dazu stammt wieder aus dem Sport. Ein spielstarkes Land wie Brasilien kann zwar in einem Fußballspiel gegen schwächere Länder verlieren, wird aber in der Regel öfter gewinnen.

Die Idee bestand nun darin, die Transitivitätsannahme der Klassifikatoren auszunutzen, indem mehr Klassen als nur die inzidenten Klassen nach einem Vergleich aktualisiert werden. Gewinnt Spieler A gegen Spieler B und hatte Spieler C zuvor gegen A gewonnen, so erhält nicht nur Spieler A einen Punkt, sondern auch C einen bestimmten Punktbetrag. Aufgrund der zunehmenden Transitivitätsbeziehungen im Verlauf des Turniers werden immer mehr Klassenbewertungen von *einem* Vergleich beeinflusst.

Konkret wird ein Bonus B eingeführt, der sich aus Punkten der Spieler ergibt, gegen die ein Spieler bisher gewonnen hatte. Diese Punkte werden durch einen sogenannten „Transitivitätsfaktor“ T gewichtet. T stellt dabei eine Funktion dar, die von k_j, $\tilde{w}_{k_j}$ und $\tilde{p}_j$ beeinflusst wird.

$$B_j = \sum_{k_i \in K, k_j \neq k_i} \tilde{W}_i * T(k_j, \tilde{W}_j, \tilde{p}_j)$$

$$\tilde{W}_j^+ = \tilde{W}_j + B_j$$

Für die Konzeption des Transitivitätsfaktors erschienen folgende Eigenschaften sinnvoll:

- Je mehr Siege eine Klasse besitzt, desto höher soll der Bonus sein, den sie erhält.
- Je mehr Spiele eine Klasse gespielt hat, desto größer ist der Transitivitätsfaktor für die jeweilige Klasse. Diese Eigenschaft basiert auf der

Überlegung, dass bei zunehmender Spielanzahl die vorläufige Punktewertung sich der tatsächlichen Punktewertung immer mehr annähert, sodass die Transitivitätsbeziehung an Wahrscheinlichkeit zunimmt.

- Jedoch soll der Bonus bzw. Transitivitätsfaktor so ausgelegt sein, dass keine falsche Dominanz entsteht. Folgendes Beispiel veranschaulicht diesen Fall:

  Beispiel 4.3 *Klasse a gewinnt $n-1$-mal und verliert einmal gegen Klasse b. Dadurch erbt Klasse b von a einen gewissen Prozentsatz der Stimmen von a, b hat keine weiteren Gewinne. Dieser gewisse Prozentsatz darf nicht dazu führen, dass die Klasse b vor a im Ranking einzuordnen ist, also $b >_\tau a$.*

Algorithmus 5 : Vererbung

```
1 begin
2     for i = 1 to maxVergleiche do
3         (Spieler_1,Spieler_2) = bestimme Paarung nach Schweizer-System
4         Gewinner = play(Spieler_1,Spieler_2)
5         Ermittle Spielermenge M, die gegen Gewinner gewonnen haben
6         Berechne Bonus für jeden Spieler aus M und addiere Bonus zu deren
          Wertungen
7 end
```

Verschiedene Versionen der Vererbungs-Erweiterung wurden an Datensätzen ausgewertet und führten zu gemischten Ergebnissen. Es konnte einerseits in einigen Fällen signifikante Verbesserungen beobachtet werden, andererseits gab es Fälle in denen keine oder negative Auswirkungen des Ranking-Errors das Ergebnis waren.

Es wird hier nicht näher auf diese Methode eingegangen, da zum einen den etablierten Feinwertungen des Schweizer-Systems (Sonneborn-Berger, Buchholz) eine höhere Priorität zugeteilt wurde und zum anderen keine effiziente Berechnung des Transitivitätsfaktors bestimmt werden konnte. Zu den Problemen, die die Konzeption einer effizienten Vererbungsmethode erschwerten, zählte die Zyklusvermeidung bei der Bonusberechnung und die Vermeidung einer falschen Dominanz. Dieser Ansatz wurde aus Gründen der Vollständigkeit hier kurz erläutert.

4.6.2 Sonneborn-Berger, Buchholz

Bei der Entwicklung der Vererbungsmethode war dem Autor nicht bekannt, dass bereits Erweiterungen für das Schweizer-System existieren. Die sogenannte Sonneborn-Berger Feinwertung ist der Vererbungsmethode sehr ähnlich.

Das Sonneborn-Berger System, von Oscar Gelbfuhs entwickelt, hatte das Ziel, Spieler mit gleicher Punktzahl im Ranking weiter zu differenzieren. Dazu nimmt man eine weitere Wertung, die *SoBerg* Wertung, für das Ranking hinzu. Die *SoBerg* Wertung eines Spielers ist bestimmt durch die Summe der vollen Punktzahlen aller Gegner, gegen die er gewonnen hatte, sowie die halbe Punktzahl von allen Gegnern, gegen die er Remis gespielt hat.

Definition 4.4 (SoBerg-Wertung, SoBerg) *Sei $P \subseteq K \times K$ die Menge der bis-*

her gespielten Paarungen. Die SoBerg Wertung S_i für Klasse k_i ist:

$$S_i := \sum_{(i,j)\in P} K_{i,j}\tilde{W}_j$$

wobei es sich hier um diskrete Klassifikatoren $K_{i,j}$ handelt:

$$K_{i,j} = \begin{cases} 0 & \textit{falls Klasse j verloren hat,} \\ 0.5 & \textit{falls Remis,} \\ 1 & \textit{sonst.} \end{cases}$$

Ein Beispiel dazu, dass aus dem Wikipedia Eintrag [6] zu Feinwertungen für Schachturniere stammt. In der Tabelle 4.3 ist das Ergebnis eines Turniers abge-

Tabelle 4.3: Am Ende eines Rundenturniers ergibt sich folgende Ergebnismatrix für die Spieler $A, B, \ldots, G$ („$\frac{1}{2}$“ = Remis, „1“ = Sieg, „0“ = Niederlage)

	A	B	C	D	E	F	G	Punkte
A	-	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1	1	5
B	$\frac{1}{2}$	-	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1	4.5
C	$\frac{1}{2}$	$\frac{1}{2}$	-	$\frac{1}{2}$	$\frac{1}{2}$	1	1	4
D	0	$\frac{1}{2}$	$\frac{1}{2}$	-	1	1	1	4
E	0	0	$\frac{1}{2}$	0	-	1	1	2.5
F	0	0	0	0	0	-	1	1
G	0	0	0	0	0	0	-	0

bildet. Spieler C und D haben die gleiche Punktzahl. Es werden nun die *SoBerg* Wertungen für beide Spieler bestimmt.

Spieler C erhält folgende SB-Punkte:

$$S_C = 0.5*5+0.5*4.5+0.5*4+0,5*2.5+1*1+1*0 = 9$$

Spieler D erhält folgende SB-Punkte:

$$S_D = 0*5+0.5*4.5+0.5*4+1*2.5+1*1+1*0 = 7.75$$

Da Spieler C die größere *SoBerg* Wertung hat, steht er im Ranking vor Spieler D.

Wie man sehen kann, haben Gewinne bzw. Remis gegen starke Klassen einen größeren Einfluss auf die Wertung als gegen schwächere. Der Remis gegen die starke Klasse A ergibt einen Punkt von 2.5, wohingegen ein Remis gegen Klasse F nur einen halben Punkt bedeutet. Beim Gegner G ist es sogar gleichgültig, ob ein Gewinn, Remis oder Verlust vorliegt, da G 0 Punkte besitzt.

Anders als bei der Vererbungsmethode werden die zusätzlichen Wertungen nicht den eigentlichen Wertungen hinzuaddiert. Sie werden lediglich in Fällen herangezogen, in denen ein Punktegleichstand mehrerer Spieler vorliegt, wobei die Wertungen nur für die entsprechenden Spieler ausgewertet werden.

Neben dem Sonneborn-Berger existiert eine weitere Feinwertung, die sogenannte Buchholz-Wertung. Diese unterscheidet sich dem Sonneborn-Berger nur dadurch, dass es keine unterschiedliche Behandlung der Punkte gibt, je nach Ausgang der Partie. Das heißt, gleichgültig ob Gewinn, Verlust oder ein Remis das Ergebnis war, werden die Punkte aller bisherigen Gegner eines Spielers summiert. Dies ist dann die Buchholz-Wertung des Spielers.

Definition 4.5 (Buchholz-Wertung) *Sei $P \subseteq K \times K$ die Menge der bisher gespielten Paarungen. Die Buchholz Wertung B_i für Klasse k_i ist:*

$$B_i := \sum_{j\in K, j\neq i} \tilde{W}_j$$

Diese Art der Wertung wird mit der Interpretation begründet, dass man bisher gegen stärkere Gegner spielen musste. Angenommen zwei Spieler besitzen die gleiche Punktzahl während eines Schachturniers. Wird die Buchholz-Wertung herangezogen, steht der Spieler mit der höheren Buchholz-Wertung vor dem anderen Spieler, da ersterer trotz stärkerer Gegner die gleiche Punktzahl wie zweiter Spieler erreichen konnte.

Auswertungen

Für diese zwei Feinwertungen wurden Auswertungen vorgenommen. Dabei wurden für jede Wertung zwei Versionen implementiert.

- **Version 1**
  Das Bewertungsschema des Schweizer-Systems bleibt gleich, das heißt für die Bestimmung der Spielstärke der Spieler wird das Verhältnis

  $$\frac{\text{Punktanzahl}}{\text{Anzahl gespielter Spiele}}$$

  von jedem Spieler herangezogen. Bei Gleichstandssituationen wird die jeweilige Feinwertung berechnet und darauf das Ranking aktualisiert. Das aktualisierte Ranking wird ausgewertet, aber nicht in der nächsten Iteration des Schweizer-Systems verwendet.

- **Version 2**
  Das Bewertungsschema wird durch die Feinwertung modifiziert. Das Schema bleibt dasselbe, außer dass für die nächste Iteration das durch die Feinwertung modifizierte Ranking als Bewertungsgrundlage verwendet wird.

Bei beiden Versionen findet eine Umsortierung von Spielern im Falle einer Punktegleichheit nur statt, sobald mindestens ein Spieler mithilfe der entsprechenden Feinwertung von den anderen Spielern diskriminiert werden kann. Die Rankings wurden nach jedem Vergleich ausgewertet.

Die erste Version entspricht der üblichen Anwendungsart der Feinwertungen. Soll während eines Turniers ein Ranking erstellt werden, so bevorzugt man in der Regel eindeutige Rankings. Diese eindeutigen Zwischenrankings werden durch die Feinwertungen möglich. Für das Schweizer-System jedoch sind Gleichstandssituationen informativer im Bezug auf die Gleichheit der Spielstärke. Die Feinwertungen entsprechen genau genommen lediglich Prognosen, die sich im weiteren Verlauf nicht unbedingt bestätigen müssen.

Die Motivation für die Auswertung der zweiten Version war die Suche nach einer Erweiterung des Schweizer-Systems, deren Ergebnis nicht zwingend identisch ist mit der Referenz - Weighted Voting. Es bestand die Frage, ob approximative Erweiterungen möglich sind, die ein besseres Verhältnis zwischen Genauigkeitsverlust und Anzahl der Spiele ermöglichen. Aus diesem Grunde wurden die modifizierten Rankings als Bewertungsgrundlage für das Schweizer-System benutzt, in der Hoffnung, aufgrund eventuell besserer Rankings, eine signifikant stärkere Konvergenz zu erzielen.

In den Abbildungen 4.12 und 4.13 sind wieder beispielhaft für die Datensätze `image` und `vowel` die Ergebnisse grafisch dargestellt.

Die Grafiken sind dabei nicht sehr aussagekräftig. Auf beiden ist hauptsächlich nur eine Kurve zu sehen. Alle Erweiterungen überdecken sich nahezu mit der

Abbildung 4.12: Datensatz `vowel` mit Feinwertungen

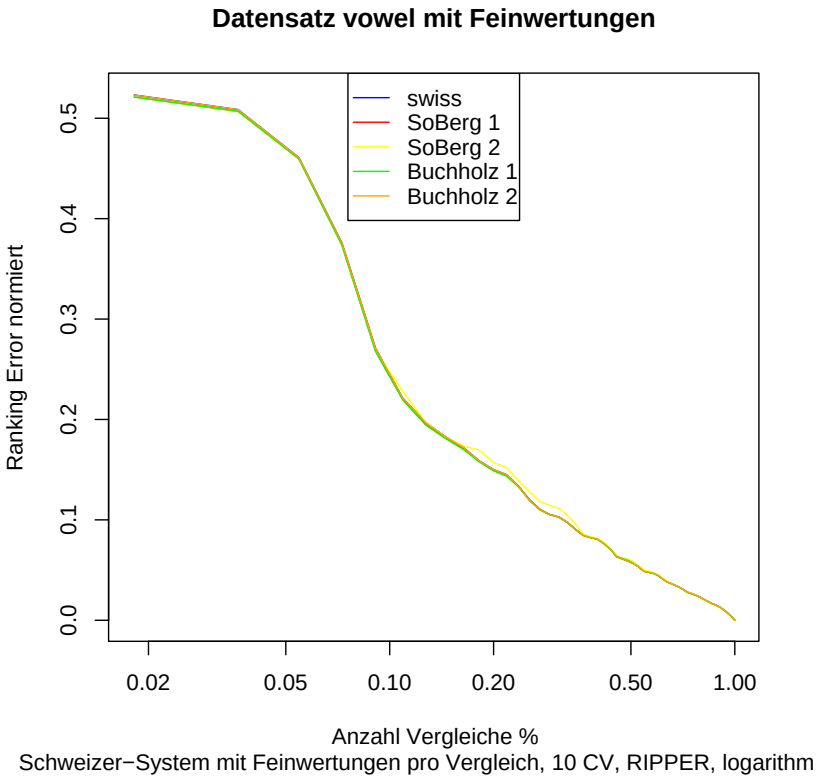
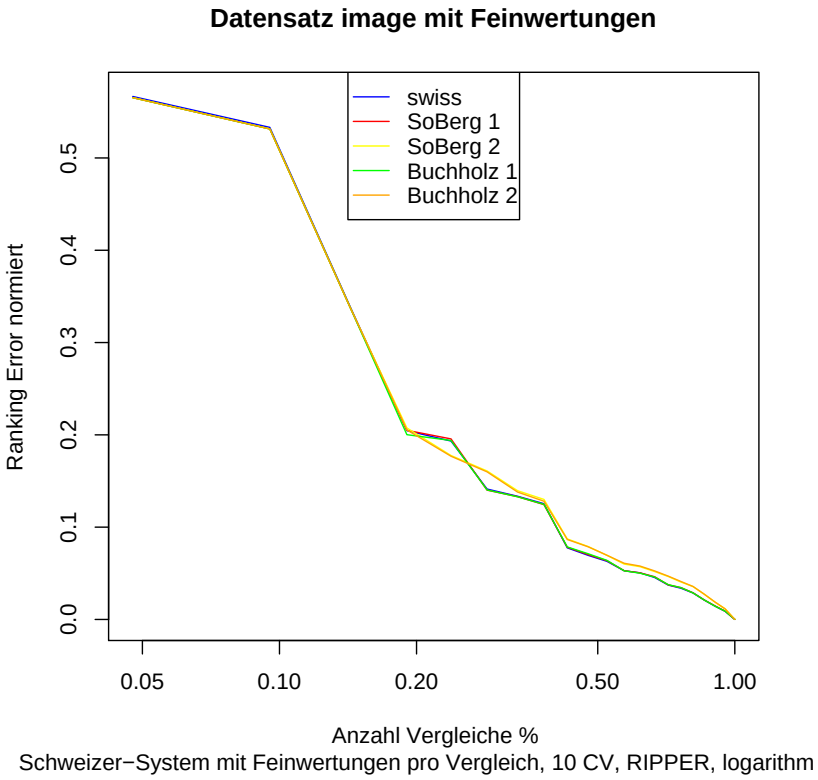


Abbildung 4.13: Datensatz `image` mit Feinwertungen



Kurve des Schweizer-Systems. Bei `image` verursacht die Buchholz-Wertung kurzzeitig eine minimale Verbesserung (grün).

Da die grafische Darstellung nicht viel Aussagekraft besitzt, wurde stattdessen eine Tabelle angefertigt.

Tabelle 4.4: Auswertungen der Sonneborn-Berger Feinwertung. Die zweite und dritte Spalte gibt den Mittelwert des Verhältnisses zwischen SoBerg 1/2 Ranking-Error zu Schweizer-System Ranking-Error wieder

Datensatz	Ratio SoBerg 1	Ratio SoBerg 2
vehicle	1 ($\pm$ 0)	1 ($\pm$ 0)
glass	1.000164612 ($\pm$ 0.000506713)	0.948780129 ($\pm$ 0.036785133)
image	1.00016504 ($\pm$ 0.00460769)	1.119807177 ($\pm$ 0.109182789)
yeast	1.012155144 ($\pm$ 0.027819882)	1.001444946 ($\pm$ 0.042759935)
vowel	0.99837131 ($\pm$ 0.004819358)	1.014541148 ($\pm$ 0.029203963)
soybean	1.002016211 ($\pm$ 0.079115186)	1.020374465 ($\pm$ 0.087806017)
letter	1.0000142 ($\pm$ 0.00012756)	1.002840891 ($\pm$ 0.007778173)

Tabelle 4.5: Auswertungen der Buchholz Feinwertung. Die zweite und dritte Spalten geben den Mittelwert des Verhältnisses zwischen Buchholz 1/2 Ranking-Error zu Schweizer-System Ranking-Error wieder

Datensatz	Ratio Buchholz 1	Ratio Buchholz 2
vehicle	1 ($\pm$ 0)	1 ($\pm$ 0)
glass	1.000066755 ($\pm$ 0.000694399)	0.948682274 ($\pm$ 0.036850418)
image	0.99769493 ($\pm$ 0.00648815)	1.119982024 ($\pm$ 0.111612066)
yeast	1.012163307 ($\pm$ 0.027793654)	1.000566501 ($\pm$ 0.043557822)
vowel	0.997842851 ($\pm$ 0.005085225)	1.015615179 ($\pm$ 0.030460505)
soybean	0.953833147 ($\pm$ 0.036558897)	1.007701388 ($\pm$ 0.044429992)
letter	0.999988525 ($\pm$ 6.69568 $\cdot 10^{-5}$)	1.002311609 ($\pm$ 0.007300357)

Die Tabellen 4.4 und 4.5 wurden folgendermaßen gebildet. Es wurden nach jedem Vergleich die Rankings mit den Feinwertungen bestimmt, darauf der Ranking-Error zur Referenz ausgewertet. Dabei wurde der durchschnittliche Ranking-Error pro Vergleichszahl in einer 10-fachen Cross-Validation gebildet. Das Verhältnis des normierten Ranking-Errors zu dem des Schweizer-Systems wurde für jede Vergleichsanzahl berechnet (Ratio). Daraus wurde schließlich der Mittelwert (Ratio) und die Standardabweichung bestimmt.

Wie man bereits aus den Abbildungen sehen konnte bestätigen die Tabellen, dass die Version 1 beider Feinwertungen nahezu keine Auswirkung auf das Ergebnis hat. Weder signifikante positive noch negative Auswirkungen sind beobachtbar. Die Ergebnisse sind im Großen und Ganzen identisch mit dem normalen Schweizer-Systems ohne jegliche Feinwertung.

Extremes Beispiel ist hierbei der Datensatz `vehicle`. Die Erweiterungen haben gar keine Auswirkungen.

Für die Version 2 der Feinwertungen lässt sich sagen, dass die Schwankungen zunehmen. Z. B. ergibt sich für den Datensatz `image` mit der Buchholz 2 Feinwertung eine relativ große Verschlechterung mit einem Mittelwert Ratio von ca. 1.12. Das bedeutet ein durchschnittlich ca. 112-prozentiger Ranking-Error des normalen Schweizer-System mit einer ungewöhnlich hohen Standardabweichung von ca. 0.11. Es gibt aber auch positive Ergebnisse wie bei dem Datensatz `glass` mit einem Mittelwert Ratio von ca. 0.95, mit einer Standardabweichung von ca. 0.03. Auch für die Version 2 der Feinwertungen lassen sich weder positive noch negative Auswirkungen bezüglich Konvergenz beobachten. Dass die Varianz zunimmt, könnte man im gewissen Masse damit begründen, dass das benutzte Bewertungsschema auf Prognosen basiert, für die keine Korrektheit garantiert ist.

Zusammenfassend sind die Auswirkungen bei beiden Versionen so gering, dass sie nahezu vernachlässigbar sind.

Erweiterte Auswertungen

Die Auswertungen erbrachten leider nicht die erhofften Ergebnisse. Es ist unverständlich, wieso die Feinwertungen in ihrer normalen Anwendungsart (Version 1) keine oder zumindest ein wenig bessere Ergebnisse erzeugen. Auf den ersten Blick erscheinen nämlich die zugrunde liegenden Ideen der Erweiterungen plausibel. Dieser Fragestellung wurde durch weitere statistische Auswertungen nachgegangen.

Folgende Informationen schienen in diesem Zusammenhang interessant:

- Anzahl der vorkommenden Gleichheitszustände - Wie oft tritt die Feinwertung in Kraft?
- Spielanzahl bei Gleichstand - Wann treten Gleichheitszustände auf?
- Anzahl der Spieler, die sich im Gleichheitszustand befinden - Was ist der Einflussbereich der Feinwertung?
- War durch Feinwertung eine eindeutige Ordnung der relevanten Spieler möglich?
- Führte die Feinwertung zu einem besseren Ranking-Error?
- Informationen über die Spielsituation - Wie viele Klassen, wie viele Klassifikatoren/gespielte Paarungen wurden für die Berechnung der Feinwertung herangezogen? Auf wie vielen Informationen basiert die Vorhersage?

Zunächst soll genauer definiert werden, was man unter „Gleichheit“ versteht.

Definition 4.6 (Gleichstandsmenge) *Eine Gleichstandsmenge $G \subseteq K$ bezüglich dem Punktewert p sei wie folgt definiert:*

$$\forall i \in G \rightarrow \tilde{W}_i = p \wedge \forall j \in K.\tilde{W}_j = p \rightarrow j \in G$$

Eine Gleichstandsmenge G bezüglich p ist somit die größte Teilmenge der Klassen K, deren Elemente $i \in G$ alle den Punktewert p besitzen.

Definition 4.7 (Gleichstand) *Besitzt der aktuelle Punktestand mindestens eine Gleichstandsmenge, so spricht man von einem Gleichstand.*

Also ist es möglich, dass in einem bestimmten Zustand des Spielverlaufs mehrere Gleichstandsmengen existieren (Gleichstandsmengen von verschiedenen Punktwerten) und somit die Feinwertungen mehr als einmal angewendet werden können. Einfaches Beispiel: 2 Spieler haben momentan 2 Punkte und 3 Spieler 1 Punkt. Somit würde die Erweiterung versuchen, zunächst die zwei Spieler mit zwei Punkten und schließlich die Gleichstandsmenge bezüglich einem Punkt weiter zu differenzieren.

In den Tabellen 4.6 und 4.7 sind die Anzahl der Gleichstände und Gleichstandsmengen der untersuchten Datensätze abgebildet. Sie geben somit die Anzahl der Spielsituationen an, in denen die Erweiterung zum Tragen kommt.

Die Werte für SoBerg 1 und Buchholz 1 sind identisch, da die Reihenfolge von der Erweiterung unbeeinflusst bleibt. Die Reihenfolge der Klassifikationsauswertungen ist in der ersten Version beider Feinwertungen allein durch das

Tabelle 4.6: Erweiterte Auswertung: Anzahl der Gleichstände für die Sonneborn-Berger Feinwertung. G steht für die Anzahl der Gleichstände und GM für Gleichstandsmengen.

	SoBerg 1		SoBerg 2	
	G	GM	G	GM
vehicle	5 (0.099%)	17 (0.335%)	5 (0.099%)	17 (0.335%)
glass	35 (0.779%)	35 (0.779%)	41 (0.912%)	41 (0.912%)
image	4494 (9.264%)	7143 (14.725%)	6251 (12.886%)	6388 (13.168%)
yeast	89 (0.133%)	89 (0.133%)	85 (0.127%)	85 (0.127%)
vowel	4482 (8.231%)	4838 (8.885%)	4360 (8.007%)	4702(8.635%)
soybean	106976 (91.595%)	259298 (222%)	106538 (91.22%)	252172 (215.9%)
letter	39105 (3.008%)	40257 (3.097%)	39089 (3.007%)	40241 (3.095%)

Tabelle 4.7: Erweiterte Auswertung: Anzahl der Gleichstände für die Buchholz Feinwertung

	Buchholz 1		Buchholz 2	
	G	GM	G	GM
vehicle	5 (0.099%)	17 (0.335%)	5 (0.099%)	17 (0.335%)
glass	35 (0.779%)	35 (0.779%)	41 (0.912%)	41 (0.912%)
image	4494 (9.264%)	7143 (14.725%)	6076 (12.525%)	6215 (12.812%)
yeast	89 (0.133%)	89 (0.133%)	85 (0.127%)	175 (0.262%)
vowel	4482 (8.231%)	4838 (8.885%)	4373 (8.031%)	4717 (8.663%)
soybean	106976 (91.595%)	259298 (222%)	106443 (91.138%)	253453 (217%)
letter	39105 (3.008%)	40257 (3.097%)	39044 (3.003%)	40196 (3.092%)

Schweizer-System bestimmt und somit entstehen die gleichen Spielsituationen.

Die Prozentangaben verdeutlichen dabei das Verhältnis der Gleichstände zu der Gesamtanzahl von Spielen bzw. ausgewerteten Klassifikatoren. Z. B. entstehen im Laufe der Version 2 der SoBerg Feinwertung für den Datensatz `yeast` insgesamt 85 Gleichstände, das heißt in 0.127% der Spiele kam es zu einer Gleichstandssituation. Bei dem Datensatz `soybean` wurden bei SoBerg 1 insgesamt 259298 Gleichstandsmengen ermittelt, das bedeutet, dass nach jedem Spiel bzw. Klassifikatorauswertung im Schnitt mehr als zwei Gleichstandsmengen in der Punktetabelle vorkamen.

Die obigen Tabellen verdeutlichen, dass außer bei dem Datensatz `soybean` die Erweiterung selten zur Anwendung kommt. Der einfache Grund ist, dass „weighted“ Klassifikatoren benutzt werden. Anders als diskrete Klassifikatoren $K_{i,j} \in \{0, 0.5, 1\}$, liefern die verwendeten Klassifikatoren bereits nach einer Auswertung minimale Differenzen, die zur besseren Unterscheidung beitragen.

Der Datensatz `soybean` stellt hier eine Ausnahme dar. Obwohl versucht wird „weighted“ Klassifikatoren zu lernen, entstehen nahezu binäre Regeln. Dies ist wohl auf eine spezielle Charakteristik des Datensatzes zurückzuführen.

Die Anzahl der Gleichstände, genauer Gleichstandsmengen, implizieren nicht automatisch eine Verbesserung des Ranking-Errors. Zunächst müssen die berechneten Feinwertungen überprüft werden, ob sie überhaupt eine eindeutige Sortierung ermöglichen. Das heißt, für zwei Klassen mit der gleichen Punktzahl könnten die ermittelten Feinwertungen für beide Klassen identisch sein, sodass keine Aussage getroffen werden kann.

Dazu kommt, dass, obwohl durch die Feinwertung eine Neusortierung möglich ist, diese nicht immer zu einer Verbesserung führt. Die Ergebnisse dazu lassen sich aus den Tabellen 4.8 und 4.9 entnehmen.

Aus diesen Ergebnissen lassen sich zwei Beobachtungen entnehmen. Zunächst einmal scheinen die Feinwertungen ihren Sinn zu erfüllen. Die Neusortierungen durch die Feinwertungen tendieren mehr oder weniger zu einer Verbesserung des Ranking-Errors. Bei dem Datensatz `image` mit Buchholz 1 ist die Anzahl der Verbesserungen minimal größer als die Anzahl der Verschlechte-

Tabelle 4.8: Erweiterte Auswertung: Auswirkungen der Feinwertungen von Sonneborn-Berger (S-Sortierung möglich, VB-Verbesserung, VS-Verschlechterung)

	SoBerg 1			SoBerg 2		
	S	VB	VS	S	VB	VS
vehicle	2	0	0	2	0	0
glass	8	0	2	8	0	2
image	2145	545	510	1869	557	280
yeast	43	10	1	39	10	1
vowel	1707	604	452	1678	630	453
soybean	236100	81056	77814	232180	84770	71600
letter	7275	2357	2610	7257	2343	2622

Tabelle 4.9: Erweiterte Auswertung: Auswirkungen der Feinwertungen von Buchholz (S-Sortierung möglich, VB-Verbesserung, VS-Verschlechterung)

	Buchholz 1			Buchholz 2		
	S	VB	VS	S	VB	VS
vehicle	2	0	0	2	0	0
glass	14	1	2	20	1	2
image	4507	812	772	3321	992	391
yeast	83	39	4	123	54	5
vowel	2541	975	564	2587	929	569
soybean	238109	99891	56817	233654	100277	54144
letter	8854	3025	2509	8793	2968	2509

rungen. Bei `soybean` hingegen ist mit der Buchholz 2 Erweiterung sogar ein 2:1 Verhältnis zu sehen.

Als zweites fällt auf, dass bei der Buchholz-Feinwertung mehr Spielsituationen sortierbar sind. Das liegt wahrscheinlich daran, dass durch die Hinzunahme von zusätzlichen Informationen (verlorene Spiele) dieser Vorteil gegenüber der Sonneborn-Berger Feinwertung ermöglicht wird. In der Tat werden durchschnittlich mehr Informationen bei der Buchholz-Feinwertung hinzugezogen als bei der Sonneborn-Berger Feinwertung, wie man beim Vergleich der Tabellen 4.10 und 4.11 sehen kann.

Es wurde zwar gesagt, dass eine Tendenz für eine höhere Anzahl an Verbesserungen besteht, jedoch darf man das Ausmaß der Verbesserungen bzw. Verschlechterungen nicht vernachlässigen. Dazu wurden für die drei Datensätze `image`, `vowel` und `soybean` in denen Gleichstände häufig vorkamen Histogramme über die Ranking-Error Veränderungen gebildet.

In der Abbildung 4.14 sind die Histogramme zu sehen. Dabei stellt die x-Achse die unnormierten Ranking-Error Veränderungen dar. Die y-Achse beschreibt die prozentuale Häufigkeit. Die Histogramme wurden über alle aufgezeichneten Statistiken der Klassifikatorauswertungen gebildet. Da der Großteil der Auswertungen keinen Gleichstand vorweiset oder trotz Gleichstand keine Sortierung möglich ist, besitzen alle Histogramme einen hohen Wert bei null. Ein negativer x-Wert beschreibt dabei eine Verbesserung des Ranking-Errors. Man kann aus dieser Grafik sehen, dass obwohl die Feinwertungen zu einer Verbesserung tendieren, die Beträge der Verbesserung und Verschlechterung sich nahezu ausgleichen.

Die Tabellen 4.10 und 4.11 bieten zwei weitere wichtige Beobachtungen. Zum einen ist die durchschnittliche Anzahl der Spieler in einer Gleichstandsmenge (*SPA*) ungefähr zwei, was die minimale Anzahl für eine Gleichstandsmenge entspricht. Das hat die Konsequenz, dass die Auswirkungen der Feinwertungen für den Ranking-Error - gleichgültig ob negativ oder positiv - relativ gering sind. Dies stimmt im Übrigen mit den Histogrammen aus 4.14 überein, die

Abbildung 4.14: Erweiterte Auswertung: Ranking-Error Veränderung durch Feinwertungen. (erste Zeile: `image`, zweite Zeile: `vowel` und dritte Zeile: `soybean`)

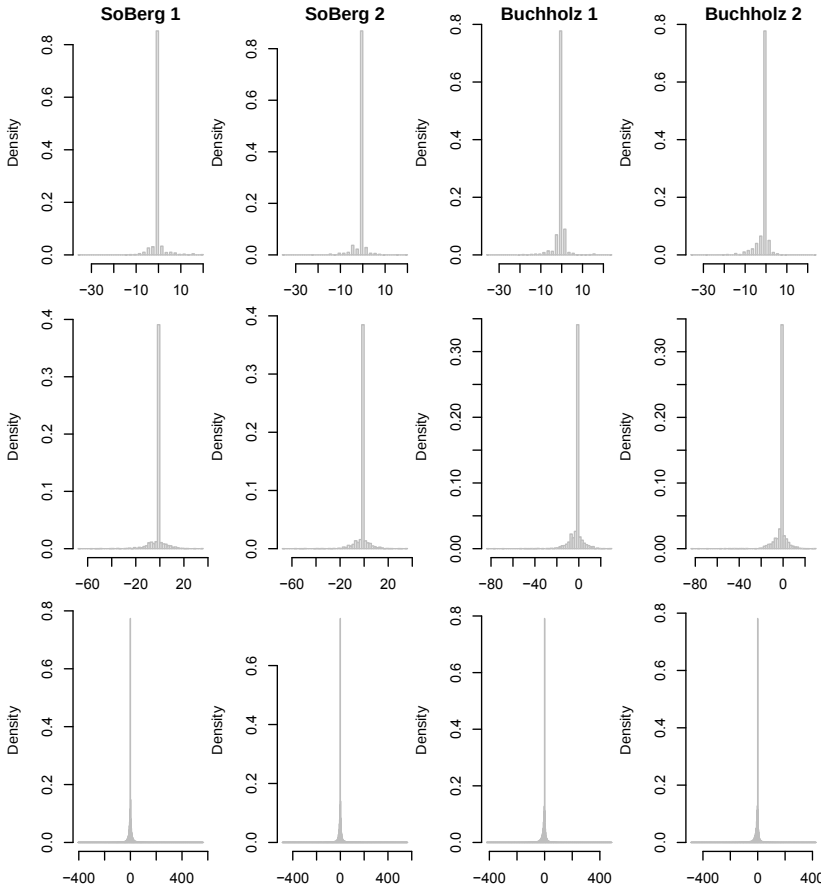


Tabelle 4.10: Erweiterte Auswertung: Einflussbereich der Feinwertungen von Sonneborn-Berger. (SPA - Anzahl Spieler in einer Gleichstandsmenge, KL - Anzahl Vergleiche/Spiele, siehe Text)

	SoBerg 1		SoBerg 2	
	durchschn. SPA	durchschn. KL	durchschn. SPA	durchschn. KL
vehicle	2	2 (33.3%)	2	2 (33.3%)
glass	2	2.75 (13.1%)	2	2.75 (13.1%)
image	2.119	3.373 (16.1%)	2.097	3.212 (15.3%)
yeast	2.022	2.465 (5.5%)	2.024	2.093 (4.7%)
vowel	2.204	3.098 (5.6%)	2.205	2.917 (5.3%)
soybean	2.912	50.833 (29.7%)	2.945	51.44 (30.1%)
letter	2.671	2.247 (0.7%)	2.671	2.240 (0.7%)

Tabelle 4.11: Erweiterte Auswertung: Einflussbereich der Feinwertungen von Buchholz. (SPA - Anzahl Spieler in einer Gleichstandsmenge, KL - Anzahl Vergleiche/Spiele, siehe Text)

	Buchholz 1		Buchholz 2	
	durchschn. SPA	durchschn. KL	durchschn. SPA	durchschn. KL
vehicle	2	2 (33.3%)	2	2 (33.3%)
glass	2	5.571 (26.5%)	2	4.35 (20.7%)
image	2.119	8.766 (41.7%)	2.099	7.58 (36.1%)
yeast	2.022	5.795 (12.9%)	2.023	5.642 (12.5%)
vowel	2.204	6.203 (11.3%)	2.205	5.551 (10.1%)
soybean	2.912	72.865 (42.6%)	2.927	72.629 (42.5%)
letter	2.671	3.340 (1%)	2.672	3.304 (1%)

eine geringe Varianz zum Nullpunkt aufweisen. Bei einer Gleichstandsmenge von zwei Spielern existieren nur zwei Möglichkeiten, Spieler A ist vor Spieler B einzuordnen oder umgekehrt. Der Einfluss einer Änderung der Sortierung auf den Ranking-Error ist dabei nach oben beschränkt durch $|2n-2|$.

Die Werte in der zweiten Spalte (*KL*) der Tabelle wurden folgendermaßen gebildet. Für jeden Spieler in der konkreten Gleichstandsmenge werden die Gegner ermittelt, gegen die gewonnen wurde oder ein Remis das Ergebnis war[4]. Von diesen Gegnern wurde die Anzahl der gespielten Paarungen aufsummiert. Mit diesem Wert sollte dargestellt werden, auf welchen Informationen, genauer gesagt aus wie vielen Spielergebnissen bzw. Klassifikationsauswertungen die Feinwertung bestimmt wird. Dies erlaubt ein Bild zu machen, wie aussagekräftig die Vorhersagen sind.

Die wichtigste Beobachtung ist, dass für die Ermittlung der Feinwertung sehr geringe Anzahlen von Klassifikatorauswertungen herangezogen werden. Z. B. beträgt für den Datensatz `vowel` (SoBerg 1) die durchschnittliche Anzahl der Klassifikatorauswertungen ungefähr 3.098. Der Datensatz `vowel` beschreibt ein 11-Klassenproblem und stellt mit der Round-Robin Binärisierung somit 55 Klassifikatoren zur Verfügung. Das bedeutet die Vorhersagen basieren im Durchschnitt auf ungefähr 5.6% der maximal möglichen Informationen - den Klassifikationsauswertungen. Dabei sind die Angaben meistens größer als die tatsächliche verwendete Anzahl von Klassifikationsauswertungen, da bei der Aufsummierung keine explizite Überprüfung auf Eindeutigkeit der Klassifikatoren vorgenommen wird.

Zusammengefasst lässt sich sagen, dass mehrere Faktoren dafür verantwortlich sind, weshalb die Feinwertungen nicht das gewünschte Ergebnis erzielen:

- **zu geringe Anzahl an Gleichstandssituationen**
  Durch „weighted“ Klassifikatoren entstehen zu selten Gleichstandsituationen. Denkbar wären Analysen mit binärisiertem Output der Klassifikatoren, jedoch ist für die Erzeugung von „weighted“ Klassifikatoren in diesem Fall keine zusätzliche Komplexität notwendig. Daneben stellen die feinen Unterschiede der Klassifikatoren wichtige Informationen dar, die nicht ohne Grund ignoriert werden sollten. Die Anzahl einer aktiven Feinwertung ist genauer betrachtet geringer als die Tabellen 4.6 und 4.7 vermuten lassen. Tatsächlich ist von diesen Gleichstandsmengen wiederum nur ein Teil verwertbar - in Fällen in denen eine eindeutige Sortierung durch die Feinwertungen möglich ist.

- **Tendenz neigt zu Verbesserungen, die Änderungsbeträge gleichen sich jedoch nahezu aus.** Die Feinwertungen führten zu einer höheren Anzahl von Verbesserungen als Verschlechterungen, wenn auch in einigen Fällen dieser Unterschied sehr gering ist. Somit scheint der plausible Grundgedanke der Feinwertungen, dass es günstiger sei, Spieler in einer Gleichstandsmenge mithilfe weiterer Spielinformationen weiter zu differenzieren, als diese in einer zufälligen Reihenfolge zu platzieren, bestätigt. Im Grunde verliert dieses Verhältnis von Verbesserungen zu Verschlechterungen jedoch seinen Wert, da die Beträge sich ausgleichen. (siehe Abbildung 4.14)

- **Einflussbereich gering - Informationsbasis gering → vage Vorhersage** Der Grund für die hohe Anzahl an Verschlechterungen durch die

[4] Für die Buchholz Wertung werden zusätzlich die verlorenen Partien berücksichtigt.

Feinwertungen und auch eventuell für einen zu *geringen* Betrag im Falle einer Verbesserung lässt sich auf die verwendeten Informationen zurückführen. Als Informationsgrundlage für die Feinwertungen wird im Durchschnitt eine sehr geringe Anzahl von Klassifikatorauswertungen benutzt, sodass man von keiner großen Vorhersage-Wahrscheinlichkeit ausgehen kann.

Es erscheint interessant, dass auch für den Datensatz `soybean` keine signifikanten Veränderungen durch die Feinwertungen sichtbar wurden, da für dieses Multiklassifikationsproblem die oben aufgeführten Gründe gar nicht oder nur sehr schwach gelten. Mit ihrer hohen Anzahl an Gleichstandsmengen, die charakteristisch für eine Auswertung von Klassifizieren mit binärem Output sind, besitzt dieser Datensatz eine Sonderstellung unter den verwendeten Datensätzen. Dieser Sachverhalt und die Evaluation der Feinwertungen anhand diskreter Klassifizierer wurde jedoch in dieser Arbeit nicht weiter verfolgt.

5 Zusammenfassung und Ausblick

In der vorliegenden Diplomarbeit wurde ursprünglich das Ziel verfolgt, eine effiziente Dekodierungsphase für die Rankinggenerierung mit paarweisen Vergleichen zu ermöglichen. Ansatzpunkt war dabei das Schweizer-System.

Neben der Implementation des Schweizer-Systems wurden Auswertungen von *Position-* und *Ranking-Errors* des Schweizer-Systems an verschiedenen Multiklassendatensätzen der UCI Repository vorgenommen. Die Ergebnisse der Auswertungen lassen sich mit folgenden Aussagen zusammenfassen:

- Ergebnisse zum Ranking-Error
  - Das Verhältnis von Genauigkeitsverlust und Vergleichsreduzierung ist nahezu konstant.
  - Das Schweizer-System ist einer zufälligen oder (Klassen-) geordneten Reihenfolge überlegen.
- Ergebnisse zum Position-Error
  - Für die Bestimmung des *top rank* sind nicht alle Vergleiche notwendig.
  - Mit zunehmender Klassenanzahl konvergiert der Position-Error schneller gegen den Nullwert.

Die Beobachtungen aus dem Position-Error waren die Motivation für die Entwicklung des *Quick (Weighted) Voting* - Algorithmus. Die Evaluation des Algorithmus auf verschiedenen Datensätzen wiesen auf eine Komplexität von $n \cdot \log(n)$ auf. Darüber hinaus wurden einige theoretische Eigenschaften des Algorithmus beschrieben und der Frage nachgegangen, in welchem Masse der Algorithmus als Suchverfahren einzuordnen ist. Durch die Abbildung des Dekodierungsproblems auf einen Graphen ließ sich eine Beziehung zu dem Algorithmus von Kruskal herstellen. Für einen zusammenhängenden, ungerichteten, gewichteten Graphen[1] kann der QWeighted Algorithmus möglichst effizient eine Obermenge des minimal aufspannenden Baumes bestimmen, wobei

[1] Es gilt zusätzlich die Bedingung, dass jeder Knoten mit jedem anderen Knoten direkt durch eine Kante verbunden ist.

die Kantengewichte nicht vorgegeben sind, sondern iterativ durch eine monotone Funktion angenähert werden.

Die allgemeine Frage, ob das Schweizer-System eine Vergleichsreduzierung für die Rankinggenerierung unter akzeptablem Genauigkeitsverlust ermöglicht, wird vom Autor verneint, da dieses Verhältnis konstant ist. Das heißt, es existiert eine lineare Abhängigkeit zwischen Vergleichsreduzierung und Genauigkeitsverlust. Diese Aussage wird unter der Annahme getroffen, dass die Komplexität der Klassifikationsauswertung je nach Basislerner gleichgestellt ist mit der Wichtigkeit eines genauen Ergebnisses. Allerdings eignet sich das Schweizer-System für Voting Dekodierungen, deren Anzahl an Klassifikationsauswertungen beschränkt werden soll, z. B. für sehr rechnerisch aufwendige Klassifikationsauswertungen. Dafür könnten die Beschränkungen durch die Vergleichsanzahl oder Rundenanzahl reguliert werden.

Zusätzlich wurden die existierenden Feinwertungen des Schweizer-Systems Sonneborn-Berger und Buchholz an denselben Datensätzen ausgewertet. Das Ergebnis war, dass weder signifikant positive noch negative Auswirkungen zu beobachten sind. Daraufhin wurden nach den Ursachen gesucht, da die Grundideen der Feinwertungen intuitiv plausibel erschienen und die Feinwertungen in realen Schach-Turnieren schon seit längerer Zeit eingesetzt werden. Dazu wurden erweiterte Statistiken auf den Datensätzen gebildet. Dabei konnte man feststellen, dass die Gründe für das Versagen der Feinwertungen aus dem Zusammenspiel mehrerer Faktoren bestehen:

- Feinwertungen kommen sehr selten zum Einsatz
  Aufgrund der verwendeten „weighted“ Klassifikatoren traten sehr selten Gleichheitszustände ein, in denen die Feinwertung herangezogen wird. Dies bedeutet wiederum, dass die Feinwertungen einen geringen Einfluss auf den Ranking-Error besitzen.
- Tendenz neigt zu Verbesserungen, jedoch gleichen sich die Änderungsbeträge aus
  Betrachtet man das Verhältnis der Verbesserungen zu Verschlechterungen der Feinwertungen bezüglich des Ranking-Errors, so ist eine positive Tendenz zu beobachten. Jedoch verliert dieses Verhältnis seinen Wert, da deren Beträge sich nahezu ausgleichen.
- Informationsbasis gering → vage Vorhersage
  Die Auswertungen ergaben, dass die zugrunde liegende Informationsbasis für eine Vorhersage im Durchschnitt sehr gering ist, sodass man von keiner hohen Vorhersagewahrscheinlichkeit ausgehen kann.

Alle Auswertungen wurden lediglich auf sechs Datensätzen der UCI Repository gebildet und größtenteils anhand eines Basislerners - dem Regellerner JRip. Die Vergrößerung der Anzahl der Datensätze und Evaluation mit mehreren Basislernern würde ein solideres Fundament für die empirischen Aussagen treffen. Jedoch wurde dies aus zeitlichen Gründen hier nicht verfolgt.

Die Auswertungen des *Quick Weighted* Algorithmus eigneten sich nur bedingt, um eine empirische Komplexitätsbestimmung zu verfolgen. Da die größte Klassenanzahl unter den verwendeten Datensätzen lediglich 26 ist, ist es schwer eine allgemeine Aussage für große n zu treffen. Hierzu könnten Analysen mit synthetisch generierten Daten weiterhelfen.

An vielen Stellen des Schweizer-Systems wurden Annäherungen des Systems, mit dem Ziel einen Kompromiss zwischen Effizienz und Genauigkeit her-

zustellen, implementiert. Das grundlegende Konzept des Schweizer-Systems lässt mehrere Variationen der Implementation zu. Es wäre interessant, diese Variationen genauer auf Komplexität und Genauigkeit zu vergleichen. Darüber hinaus wäre eine Evaluation der Feinwertungen mit binärisierten Klassifizierern denkbar.

Literaturverzeichnis

[1] L.Breiman et. al. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[2] K. K. Chin. *Support Vector Machines applied to Speech Pattern Classification.* Master's thesis, Engineering Department, Cambridge University, 1999.

[3] W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. *In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99)*, 335–342, Menlo Park, CA, 1999. AAAI/MIT Press.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest. Greedy Algorithms. *Introduction to Algorithms*, 329–355, MIT Press, 1990.

[5] F. Cutzu. Polychotomous Classification with Pairwise Classifiers: a New Voting Principle. *IUCS Technical Report 573*, Jan 2003.

[6] Artikel „Feinwertungen für Schachturniere". In *Wikipedia: Die freie Enzyklopädie.* Stand: 29. Oktober 2006. http://de.wikipedia.org/w/index.php?title=Feinwertungen_f%C3%BCr_Schachturniere&oldid=23138972 (abgerufen am 15. November 2006).

[7] N. Friedman, D. Geiger and M. Goldszmidt. Bayesian network classifiers. *Machine Learning* 29 (2), 131–163, 1997.

[8] J. Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[9] S. Har-Peled, D. Roth and D. Zimak. Constraint classification: A new approach to multiclass classification. In N. Cesa-Bianchi, M. Numao, and R. Reischuk, editors, *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT-02)*, 365–379, Lübeck, Germany, Springer, 2002.

[10] E. Hüllermeier and J. Fürnkranz. Comparison of Ranking Procedures in Pairwise Preference Learning. *IPMU-04, International Conference on*

Information Processing and Management of Uncertainty in Knowledge-Based Systems. Perugua, Italy, 2004.

[11] E. Hüllermeier and J. Fürnkranz. Pairwise Preference Learning and Ranking. *Machine Learning: ECML 2003* 145–156. Springer Berlin / Heidelberg, 2003.

[12] T. Ihringer. Graphentheorie, Kombinatorische Optimierung (Minimale aufspannende Wälder). *Diskrete Mathematik.* 11–19, 86–88, B.G.Teubner Stuttgart, Leipzig, 1999.

[13] R. M. Karp. Reducibility among combinatorial problems. In RE Miller and JW Thatcher, editors, *Complexity Of Computer Computations*, 85–103. New York: Plenum Press, 1972.

[14] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives; Preferences and Value Tradeoffs.* John Wiley & Sons, 1976.

[15] U. H.-G. Kreßel. Pairwise classification and support vector machines. *Advances in Kernel Methods: Support Vector Learning*, 255–268, The MIT Press, Cambridge, MA, 1999.

[16] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[17] D. J. Newman, S. Hettich, C. L. Blake and C. J. Merz. *UCI Repository of machine learning databases.* http://www.ics.uci.edu/~mlearn/MLRepository.html. University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[18] J. Platt, N. Cristianini and J. Shawe-Taylor. Large Margin DAGS for Multiclass Classification. In *Advances in Neural Information Processing Systems 12*, 547–553, The MIT Press, 2000.

[19] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[20] T. L. Saaty. *The Analytic Hierarchy Process.* McGraw-Hill, NewYork, 1980.

[21] B. Schölkopf and A. Smola. *Learning with Kernels.* MIT Press, 2001.

[22] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 48:303–312, 1961.

[23] F. Takahashi and S. Abe. Optimizing directed acyclic graph support vector machines. *Artificial Neural Networks in Pattern Recognition (ANNPR 2003)*, 166–170, September 2003.

[24] G. I. Webb, J. Boughton and Z. Wang. Not so Naive Bayes: Aggregating one-dependence estimators. *Machine Learning* 58(1), 5–24, 2005.

[25] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.

[26] World Chess Federation - Federation Internationale des Echecs. FIDE Swiss Rules. *FIDE Handbook*, chapter C.04 http://www.fide.com/official/handbook.asp?level=C04 (abgerufen am 15. November 2006).

A Verschiedene Anhänge

A.1 Auswertungen zum Ranking-Error

Abbildung A.1: Ranking-Error Auswertungen für `vehicle` und `glass`

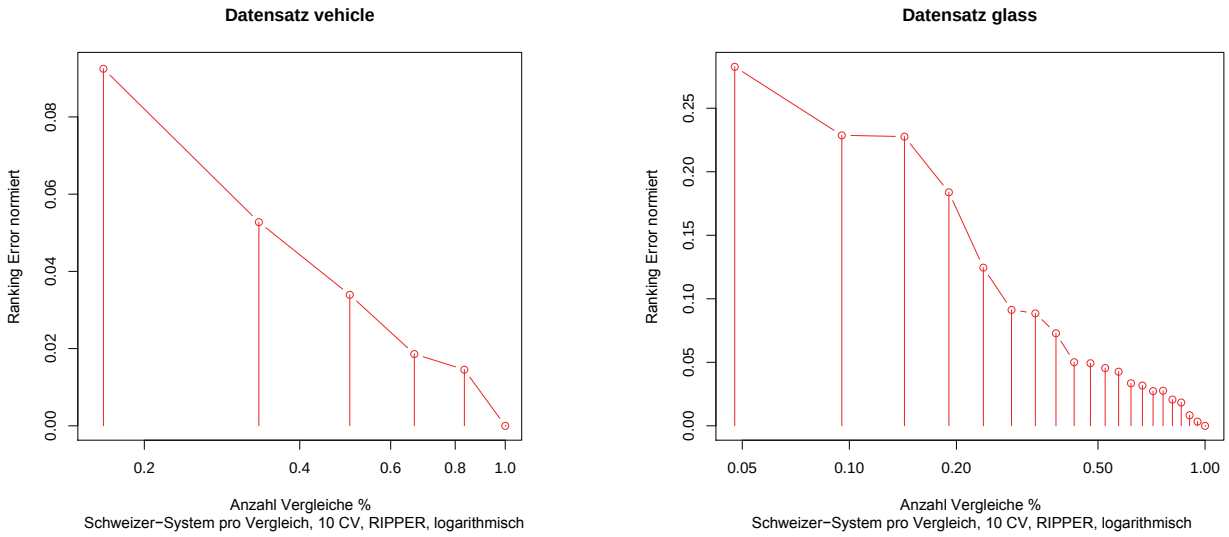


Abbildung A.2: Ranking-Error Auswertungen für `image` und `yeast`

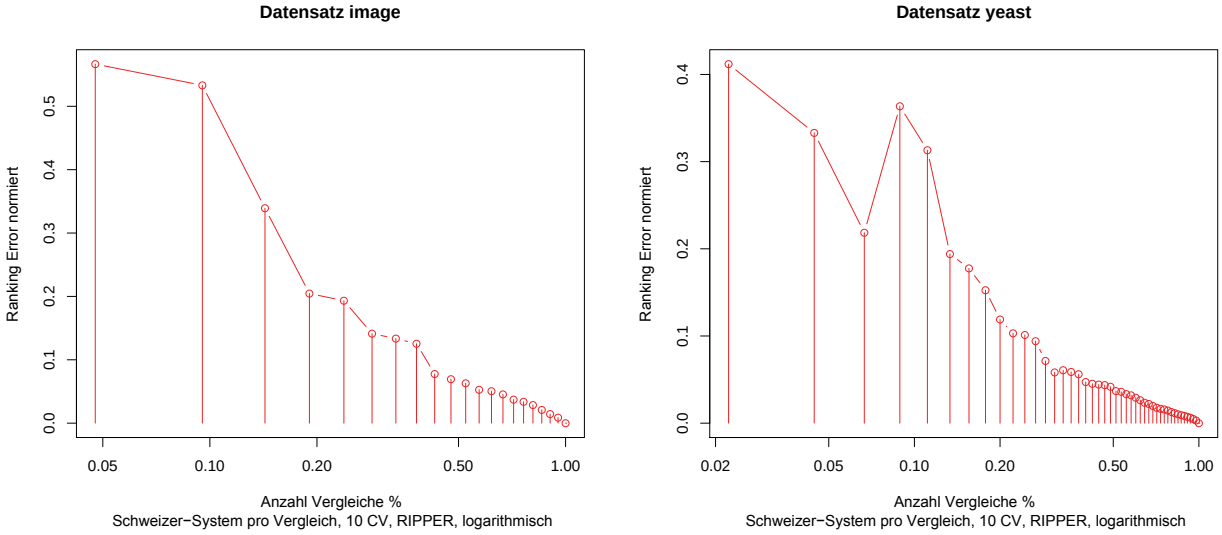


Abbildung A.3: Ranking-Error Auswertungen für `vowel` und `soybean`

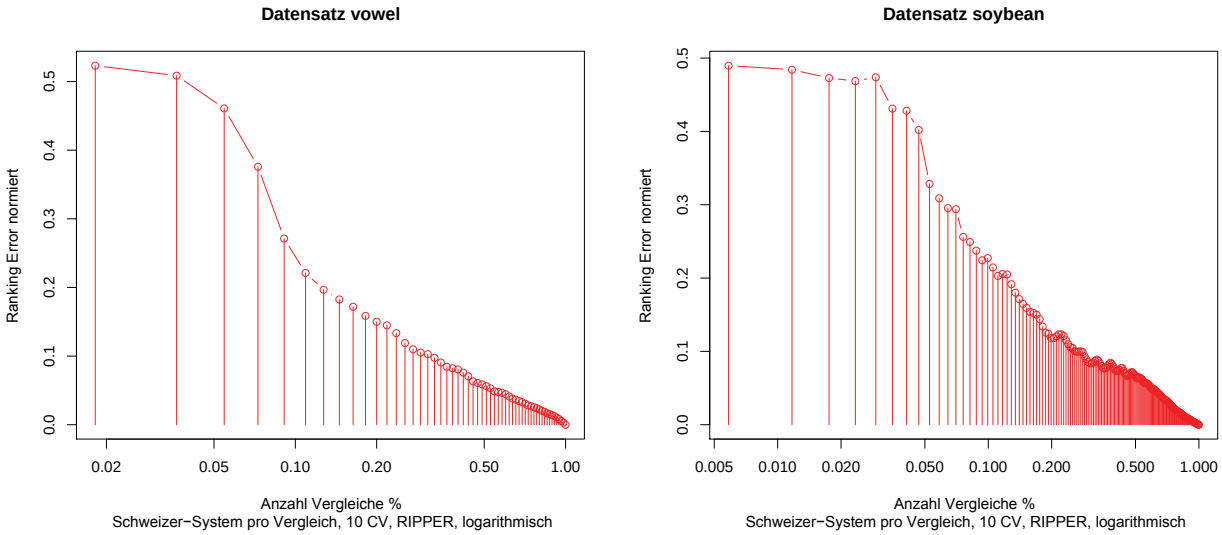
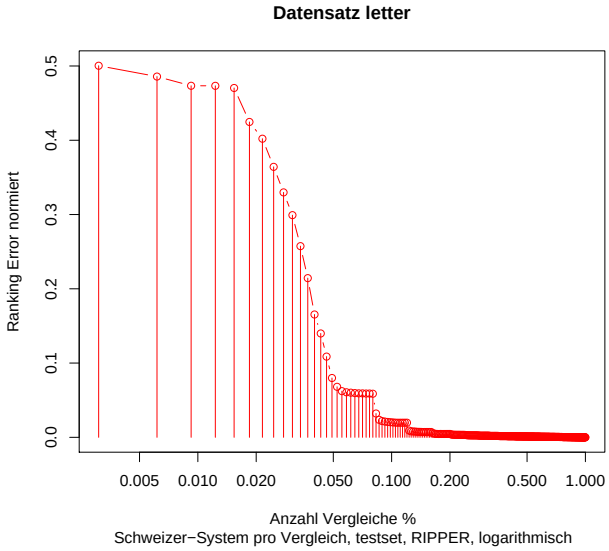
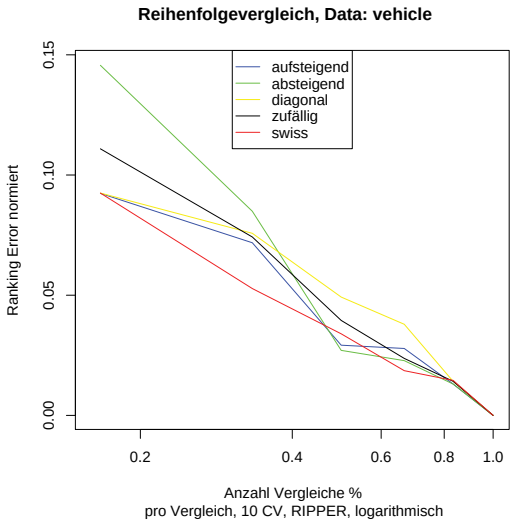


Abbildung A.4: Ranking-Error Auswertungen für `letter`



A.2 Schweizer-System vs. andere Reihenfolgearten

Abbildung A.5: Reihenfolgevergleich für `vehicle` und `glass`



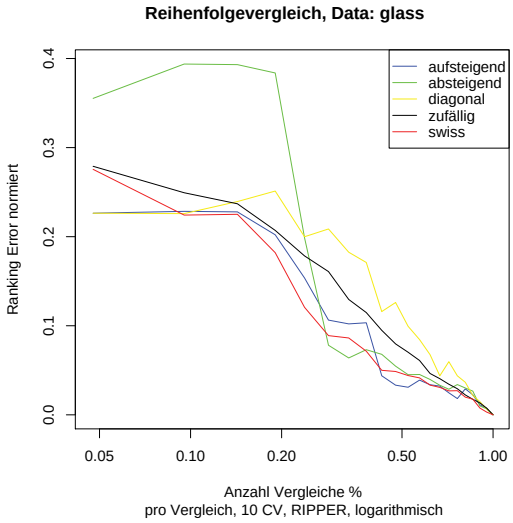
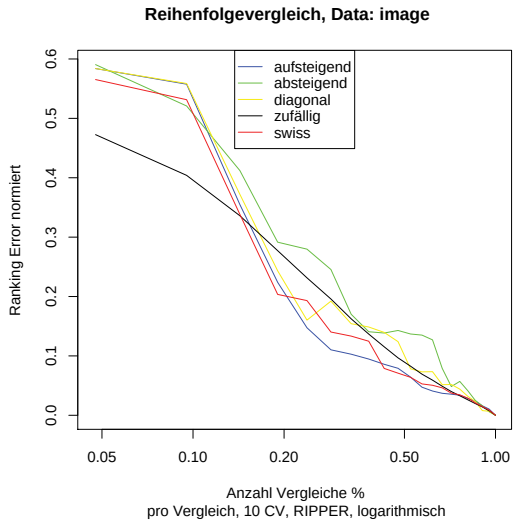


Abbildung A.6: Reihenfolgevergleich für `image` und `yeast`



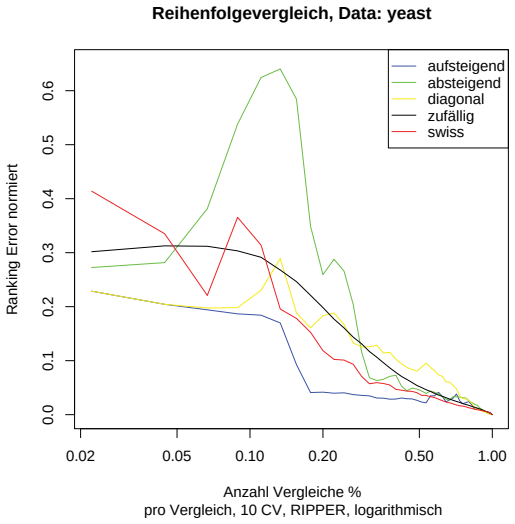
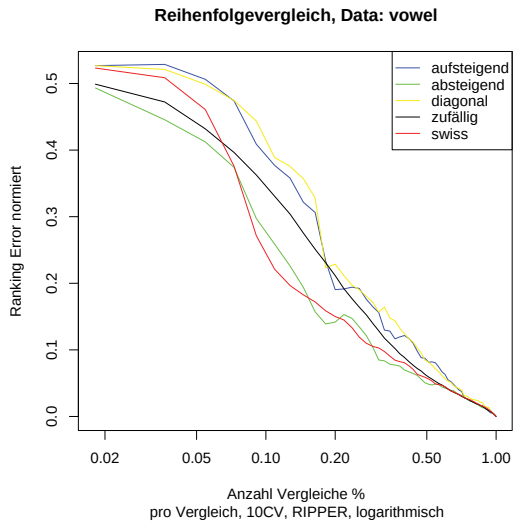


Abbildung A.7: Reihenfolgevergleich für `vowel` und `soybean`



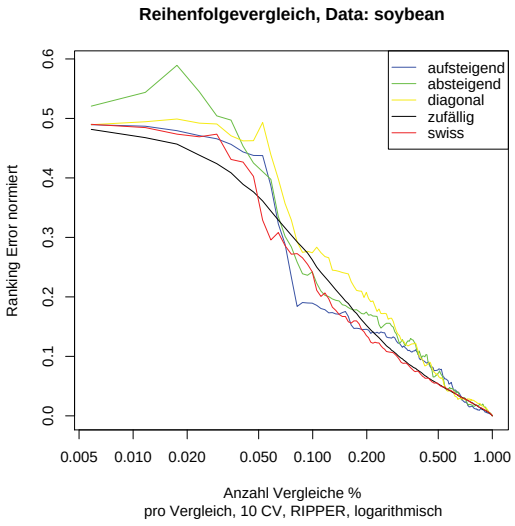
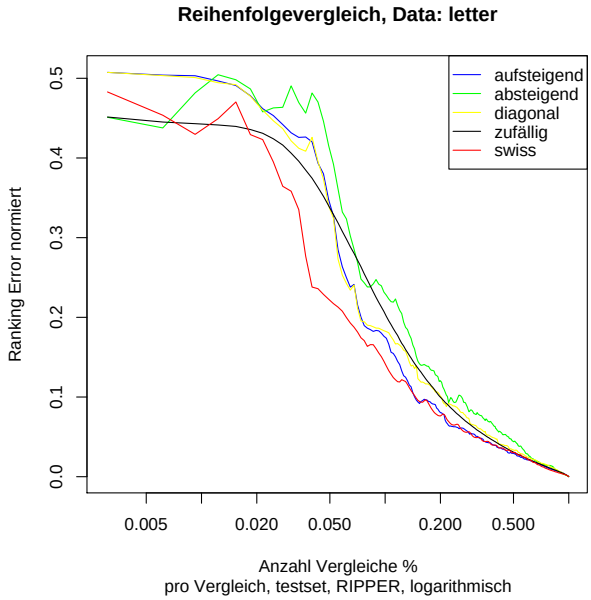
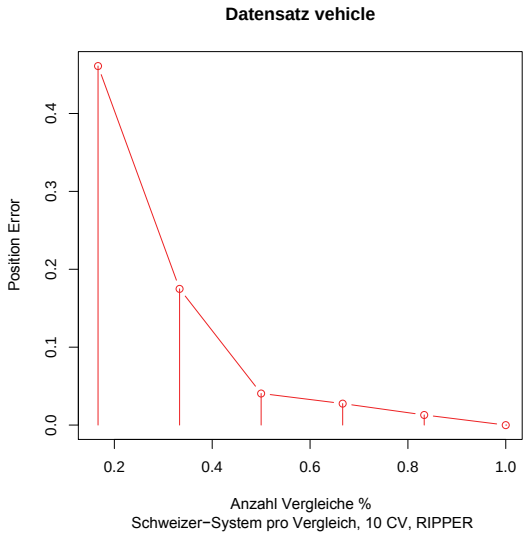


Abbildung A.8: Reihenfolgevergleich für `letter`



A.3 Auswertungen zum Position-Error

Abbildung A.9: Position-Error Auswertungen für `vehicle` und `glass`



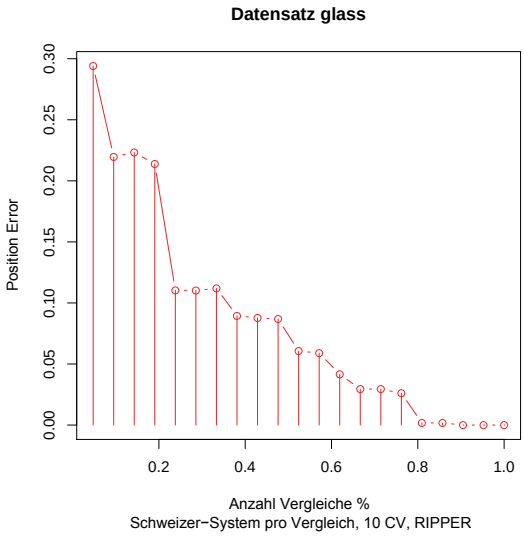


Abbildung A.10: Position-Error Auswertungen für `image` und `yeast`

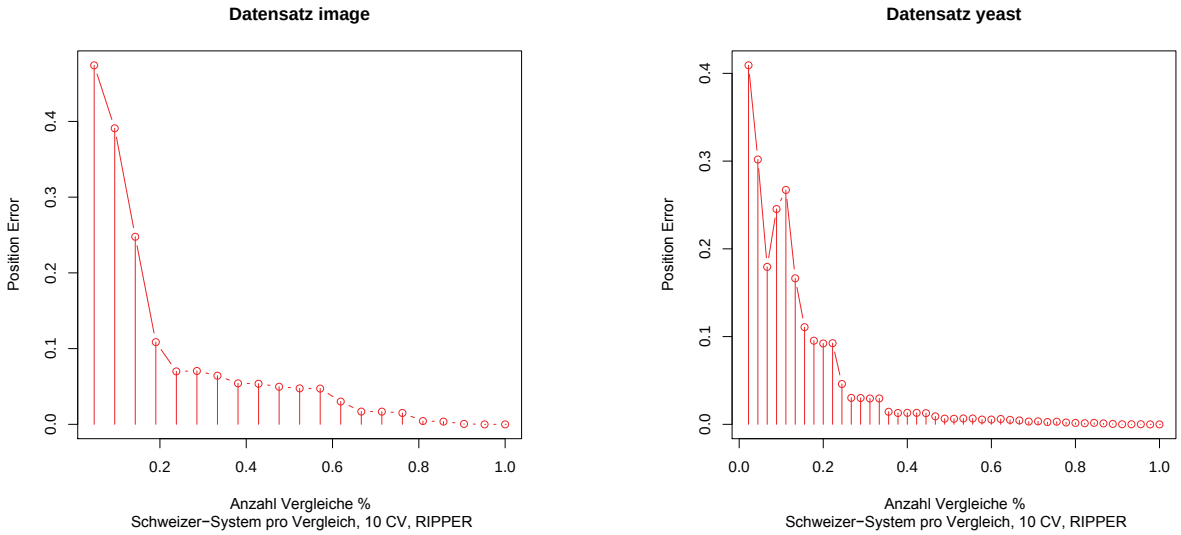


Abbildung A.11: Position-Error Auswertungen für `vowel` und `soybean`

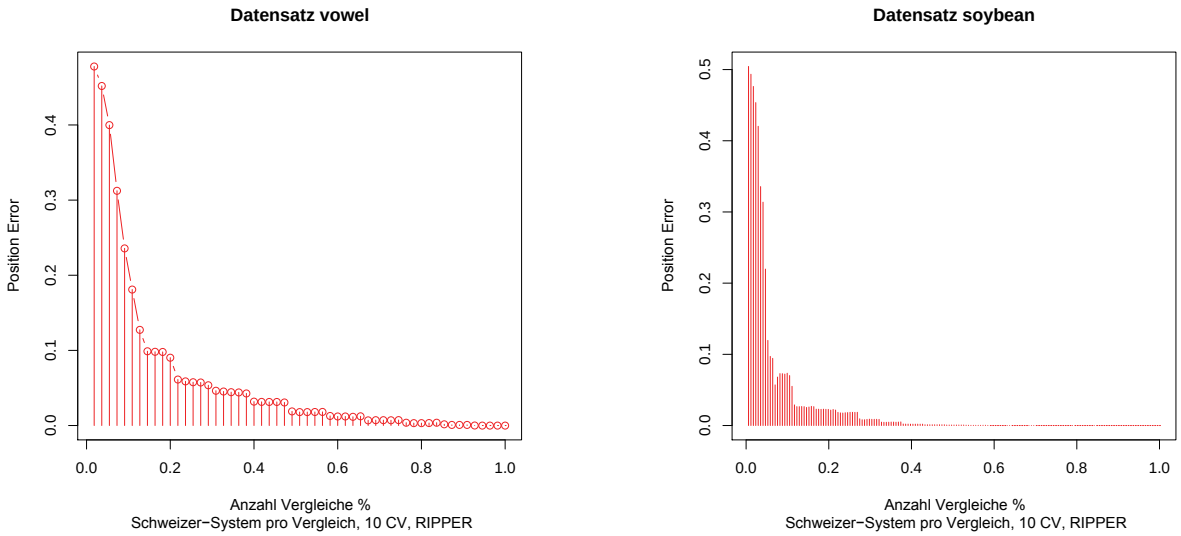
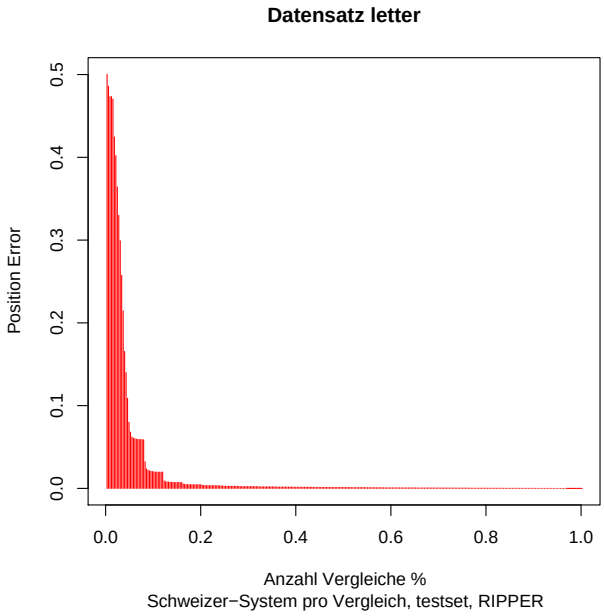
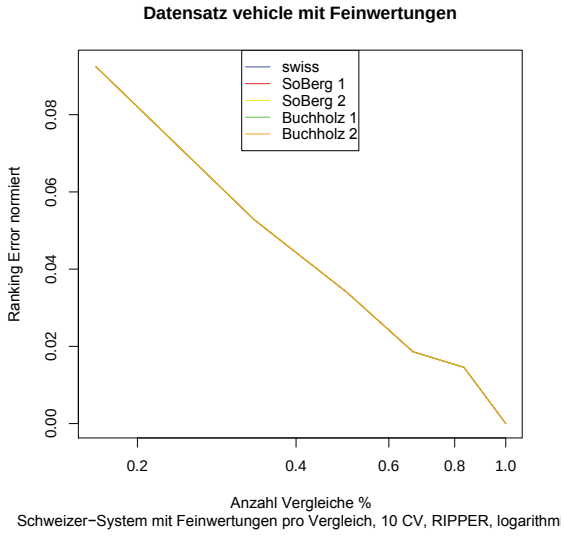


Abbildung A.12: Position-Error Auswertungen für `letter`



A.4 Auswertungen zu Feinwertungen

Abbildung A.13: Auswertungen der Feinwertungen für `vehicle` und `glass`



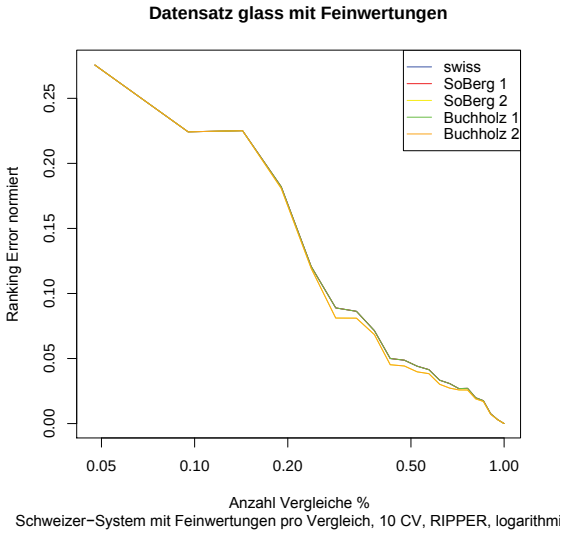


Abbildung A.14: Auswertungen der Feinwertungen für `image` und `yeast`

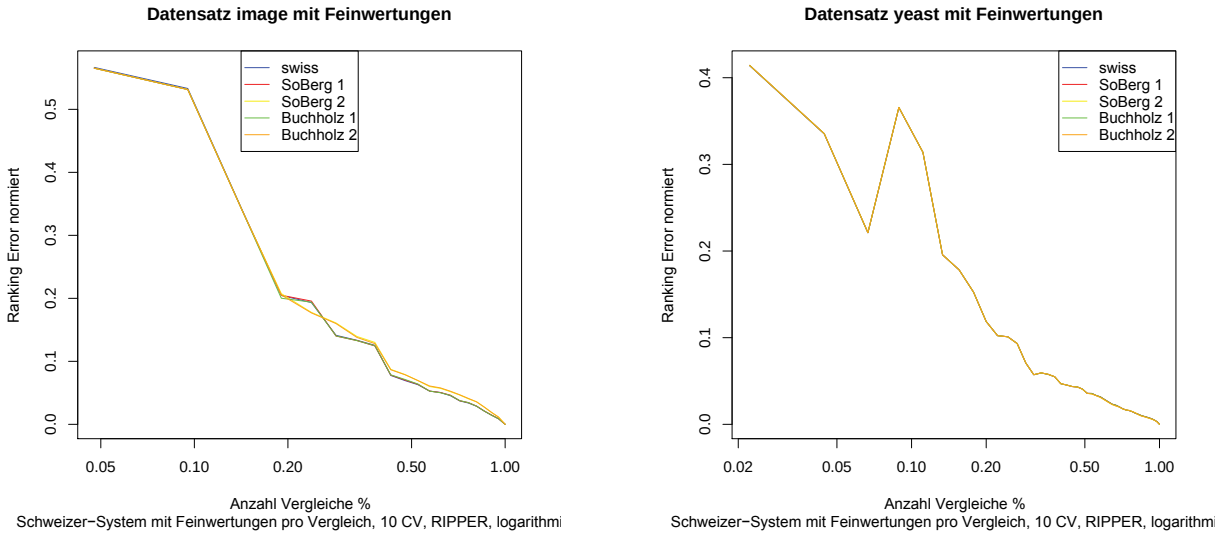


Abbildung A.15: Auswertungen der Feinwertungen für `vowel` und `soybean`

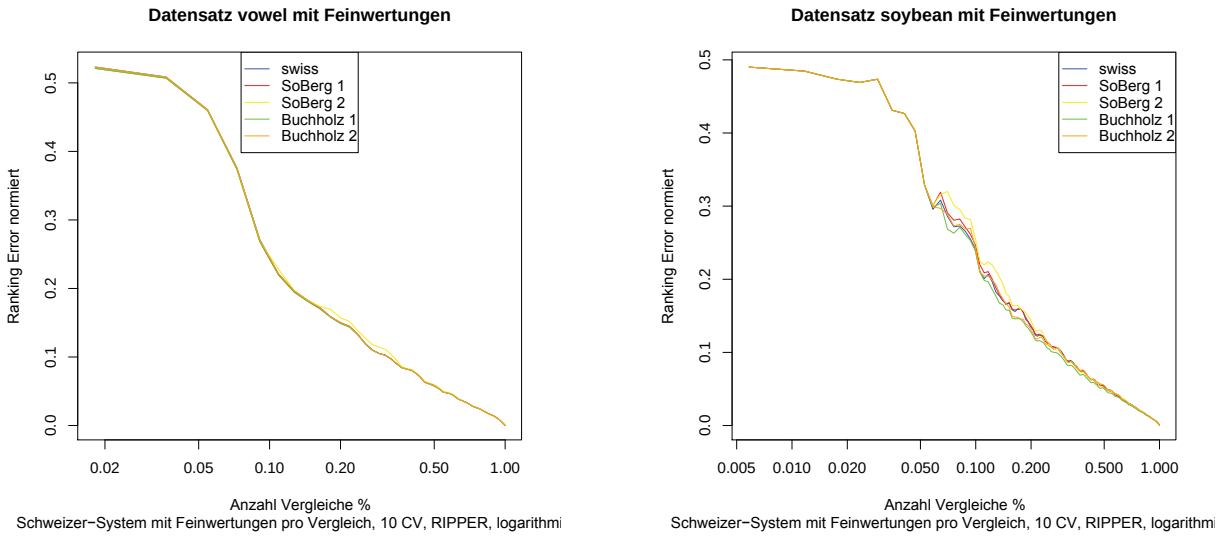


Abbildung A.16: Auswertungen der Feinwertungen für `letter`